

© 2009 Daniel Thomas Rock

ADAPTATION OF THE C.H.A.D. COMPUTER LIBRARY TO NUCLEAR SIMULATIONS

BY

DANIEL THOMAS ROCK

B.S., University of Cincinnati, 1997

M.S., University of Illinois at Urbana-Champaign, 2001

DISSERTATION

**Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Nuclear Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2009**

Urbana, Illinois

Doctoral Committee:

**Professor Rizwan Uddin, Director of Research, Chair
Professor Barclay G. Jones
Professor Clifford E. Singer
Associate Professor, Magdi Ragheb
Dr. Constantine Tzanos, Argonne National Laboratory**

ADAPTATION OF THE C.H.A.D. COMPUTER LIBRARY TO NUCLEAR SIMULATIONS

Daniel Thomas Rock, Ph.D.
Department of Nuclear, Plasma, and Radiological Engineering
University of Illinois at Urbana-Champaign, 2009
Dr. Rizwan Uddin, Advisor

The Computational Hydrodynamics for Automotive Design computer program, CHAD, is a modern, three-dimensional computational fluid dynamics code that holds promise for fulfilling a need in the nuclear industry and academia. Because CHAD may be freely distributed to non export controlled countries, it offers a cheap and customizable CFD capability.

Several modifications were made to CHAD to make it more usable to those in industry and academia. A first order up-winding scheme for momentum and enthalpy and a reformulated continuity equation were migrated from a historical version of CHAD developed at Argonne National Laboratory. The Portable, Extensible Toolkit for Scientific Computing, PETSc, was also added as an optional solver package for the original and reformulated continuity equations. PETSc's highly optimized parallel solvers can be activated from either CHAD's input file or the command line. Solution times for PETSc based calculations depend in large part on convergence criteria provided, however improvements in CPU time of approximately one-third have been observed.

CHAD was further extended by adding a capability to monitor solution progress by specifying a coordinate in space, as well as monitoring the residuals in the problem. The

ability to model incompressible fluids was also added to the code. Incompressible fluid comparisons were made using several test cases against the commercial CFD code Fluent and found to agree well.

A major limitation of CHAD in the academic environment is a limited mesh generation capability. A tool for CHAD was developed that translates Gambit based neutral mesh files into a CHAD usable format. This tool was used to translate a large mesh representing a simplified cooling jacket of a BWR control rod drive. This model serves as a practical, demonstration application of a nuclear application for CHAD and PETSc. Both CHAD with PETSc and Fluent were used to obtain solutions to this problem. The overall agreement between the two codes is good, though both applications could benefit from larger meshes.

Finally, the documentation in the public domain regarding CHAD is very limited. Therefore, a User's Manual was written for CHAD that is designed to reduce the time required to bring a new CHAD user to a productive level.

for Ting

Acknowledgements

This dissertation is the result of a body of work that surely would not have been possible if not for the support of a number of important people. From the start, my advisor, Dr. Rizwan-uddin has led me into a field in which I initially knew nothing about, but now have a profound interest. In taking my journey through the academic world, he has provided significant advice and friendship. Without him, I would not be employed doing things I love.

None of this would be possible, of course, if it were not for Dr. David Weber and Dr. Constantine Tzanos from Argonne National Lab. They funded a significant portion of my research, and provided me with this thesis topic, one that has been both challenging and highly educational. The use of their reactor engineering computing cluster made portions of this dissertation possible.

My officemates and coworkers have also provided significant advice and support during the years of this dissertation. Yizhou Yan and Jin Yan have been instrumental in helping me with issues regarding Fluent, and have provided substantial advice in generating quality meshes and models. Others often provided advice and assistance on more general but nonetheless just as important matters, including Jianwei Hu, Suneet Singh, Prashant Jain and Quan Zhou. Brad Wescott was a particularly good friend during my studies and we regularly discussed our research and other interesting scientific topics.

Outside of academia, my parents were always there to encourage me when things just didn't seem to go right. And Yi-wei Chen's blueberry muffins played a significant role in keeping me both sane and well fed.

Without any of you, the staff at UIUC, other family, and all my friends who have offered support, this would not have been possible. You all have my deepest gratitude.

Table of Contents

List of Acronyms and Abbreviations.....	x
Definition of Symbols.....	xi
1. Introduction	1
1.1 Background.....	1
1.2 Motivation	2
1.3 Objective.....	6
2. The C.H.A.D. Code	10
2.1 C.H.A.D. (Computational Hydrodynamics for Advanced Design)	10
2.2 CHAD and the Three Steps of CFD Analyses.....	13
3. Revisions to the CHAD Continuity Solver	29
3.1 The CHAD Continuity Solver	29
3.2 The CHAD Continuity Equation	30
3.3 The Reformulated Continuity Equation	33
3.4 Different Versions of the Continuity Equation.....	36
3.5 Solving the Continuity Equation Using GCR.....	41
4. Implementing PETSc Into CHAD	47
4.1 An Overview of PETSc.....	47
4.2 The Matrix Equations.....	50
4.3 Final Convergence and Variable Updates	55
5. Programming PETSc Into CHAD	57
5.1 Preparing CHAD for PETSc.....	57
5.2 Initializing PETSc.....	59
5.3 Loading Data into PETSc.....	64
5.4 Solving the Matrix System and Extracting Results	69
5.5 Terminating PETSc.....	73
6. Testing the PETSc Implementation	76
6.1 Testing Overview	76
6.2 The Computing Environment	79
6.3 Solution Monitoring and Convergence Controls in CHAD	80
6.4 Solutions to the Lid-Driven Cavity Problem.....	90
6.5 Computational Controls in PETSc.....	107
6.6 Solution Times using CHAD and PETSc.....	114
7. Incompressible and Buoyant Flows.....	130
7.1 Equations of State in CHAD	130

7.2	Adding an Incompressible Equation of State	130
7.3	Buoyant Flows	134
8.	A Practical Nuclear Application Solved Using CHAD and PETSc	161
8.1	A Practical, Nuclear Design Application	161
8.2	CRD Model Results	168
8.3	Discussion of the CRD Model Results.....	179
9.	Conclusions and Future Work.....	185
9.1	Conclusions	185
9.2	Future Work.....	191
	References	194
	Appendix A: Communication in CHAD	205
A.1	Introduction.....	205
A.2	Nodes.....	205
A.3	Connections	206
A.4	Gather Operations	208
A.5	Scatter Operations	209
	Appendix B: The Gambit to CHAD Mesh Translator.....	211
B.1	Introduction.....	211
B.2	Rules for Generating Meshes Destined for CHAD.....	212
B.3	Gambit Neutral Mesh Format Translation Procedure	213
B.4	The Mesh Translator Source Code.....	219
	Appendix C: CONTAB	232
C.1	Velocity Constraints.....	232
	Appendix D: The ACG Group CHAD User Guide.....	234
	Appendix E: Installation and Configuration of Systems to Run CHAD	238
E.1	Software Overview	238
E.2	Software Installation Tips	241
E.3	Running CHAD Sample Problems	251
E.4	Creating Models in CHAD	251
E.5	The CHAD Input File	260
E.6	Boundary Conditions in CHAD	269
E.7	Running CHAD	276
E.8	Post-Processing CHAD.....	278
E.9	Internals of CHAD	280
	Appendix F: Source Code Modifications	292
F.1	Diff of CHAD_ANL to CHAD_UIUC	293
F.2	Diff of CHAD 5.0 to CHAD_UIUC	302

F.3 Modified and New Source Routines	312
Author's Biography	444

List of Acronyms and Abbreviations

ANL	Argonne National Lab
BWR	Boiling Water Reactor
CAD	Computer Aided Design
CFD	Computational Fluid Dynamics
CHAD	Computational Hydrodynamics for Advanced/(Automotive) Design
CRD	Control Rod Drive
NO-UTOPIA	Node-Centered Unstructured Topology, Parallel Implicit Advection
PETSc	Portable Extensible Toolkit for Scientific Computing
PGS	Pressure Gradient Scaling
SIMPLE	Semi Implicit Method for Pressure Linked Equations

Definition of Symbols

a	Pressure-Gradient-Scaling Parameter
d	half the distance between two nodes
\mathbf{I}	identity tensor
K	specific turbulence kinetic energy
\mathbf{n}	unit outward normal to surface of control volume V
\mathbf{N}	an optional, explicit velocity node-coupler
p	hydrostatic pressure
Re	Reynolds Number
S	surface of control volume V
t	time
T	temperature
\mathbf{u}	fluid velocity
\mathbf{v}	control volume velocity
VFX	Upwind direction indicator
α	median mesh face
μ	first coefficient of viscosity
v	node
$v\alpha$	node connected to node v by the edge touching face α
ρ	density
σ_T	turbulence switch
τ	stress tensor

Chapter 1: Introduction

1.1 Background

The modern age of computing has revolutionized the design, analysis, and optimization of modern machinery. The resurgence of nuclear power has renewed interest in, among other areas, using modern computing capabilities to solve thermal-hydraulic problems in ways that have thus far proven too costly or too computationally intensive to solve. These challenges in reactor thermal-hydraulics exist because in addition to the extreme fluid and temperature conditions found inside reactors, especially boiling water reactors, the reactor cores are geometrically complex. They contain large numbers of very small geometric entities that exert substantial effects on the flow field that affect a plant's operation. For example, *mixing vanes*—tiny twisted bits of metal found on spacer grid plates—induce turbulence and promote mixing between fuel rods, affecting core pressure drops, material deposition on fuel rods, and void profiles, among other things. The computer modeling of but a single mixing vane requires a large number of computational points in its proximity. Thus, the detailed simulation of even a few fuel channels in a reactor core requires substantial computational capabilities.

Computational Fluid Dynamics (CFD) can provide detailed information about the thermo-fluid conditions inside reactors. Early generations of CFD software were invariably limited in their ability to solve large and complex problems by the hardware capabilities of early computers, as well as by the numerical methods employed. In practice, these early codes were limited to modeling only one or two spatial dimensions

on problems of relatively simple geometric complexity. Because of these limitations, large and complex simulations such as those presented by mixing vanes were essentially impossible. As a result, the data required for design and optimization came from experiments. These experiments require costly facilities and are expensive to operate.

The past ten years have witnessed major advances in the computational ability to model thermo-fluid flow. Recent generations of CFD codes have fared significantly better at solving larger and more complex problems, due in no small part to major advances in both desktop and parallel computing (Strohmaier, et al., 2005). Efforts have already begun to model spacer grids and mixing vanes using CFD (Anglart 1997, Chun, 1998, and Grunwald, 2002). But because there may be over 10,000 such mixing vanes in a reactor core, and because there are thousands of fuel channels (subchannels), the detailed modeling of an entire reactor core is generally considered a near impossibility, even with the most powerful supercomputers currently available (Weber, 2000). Moreover, as most of the recent CFD codes are commercial, they are expensive. Additionally, as the source code for commercial CFD codes is not commonly available, users often cannot completely customize their code. It is, at least partly, for these reasons that modeling nuclear reactor cores using CFD codes still remains a challenge to engineers.

1.2 Motivation

If a CFD code is general enough to handle the difficulties associated with the meshes of complex geometries, yet robust enough to be used on a variety of parallel

computers or parallel computing clusters, and capable of numerically handling the conditions inside a nuclear reactor, then it may be possible to effectively model problems of interest to the nuclear industry that have thus far proved impossible or prohibitively costly. For example, while it may still be nearly impossible to model an entire reactor core using a CFD code, it would nevertheless be highly useful to model an entire fuel bundle to a high level of detail. Such a capability would provide a significant increase in value of the CFD code. Furthermore, if such a CFD code were available in the public domain or freely distributed to government labs or universities, and rivaled expensive commercial CFD products in capability and performance, it would provide an important tool to the nuclear engineering community. Availability of source code would encourage development of other nuclear specific features in the CFD domain, such as the ability to predict critical heat flux (CHF) via CFD, model two-phase flow, or perform safety analyses (Scheuerer et al., 2004). The applications that could benefit from CFD are not limited to the general topics just listed. Specific applications for CFD might include detailed analyses of pressure drops across channels, acoustics and vibrational analyses in steam dryers, jet pump flows, conditions below lower tie plates, steam line manifolds, CRUD deposition, dry cask storage, core spray analyses, and Loss Of Coolant Accidents (LOCA),

The CHAD code (*Computational Hydrodynamics for Advanced / Automotive Design*) holds promise for meeting some of the demanding requirements of CFD codes dictated by many nuclear engineering applications. Written for the automotive industry with a focus on combustion and shock wave applications, it has many of the capabilities

typically found in modern commercial CFD codes. With several improvements, it could become a viable CFD code for use in nuclear applications.

In its latest incarnation, CHAD has a number of characteristics that make it appealing for use by the nuclear industry. First, CHAD uses unstructured meshes. This facilitates the meshing of complex geometries, such as mixing vanes in a reactor core. Second, CHAD is portable on a variety of parallel computer architectures, and therefore likely usable on whatever in house computing clusters are available. Third, CHAD is essentially free (subject to export control regulations). This is a major advantage of CHAD over commercial CFD codes, especially within academia where the cost of licenses for multiple processors, support, and maintenance fees can be prohibitively expensive. Finally, the source code for CHAD is freely available. This allows for further development of the code to meet customized applications.

Despite its attractive characteristics, several shortcomings of CHAD have prevented its emergence in nuclear applications, as well as its overall use. One shortcoming is the speed of the code. It has been suggested that the continuity equation solver is responsible for a substantial part of this problem. Our experience confirms that the continuity equation requires approximately 35% of the total solution time, making it a prime candidate for making substantial improvement in solver speed. Currently, CHAD uses the Generalized Conjugate Residual solver (GCR) (Elman, 1982) which is a variant of ORTHOMIN (Vinsome, 1976, Greenbaum, 1982) to obtain solutions to the momentum, energy, and continuity equations.

Another shortcoming of CHAD is its inability to reduce residuals to convergence values at corner and wall nodes. This is because in CHAD, some boundary conditions are applied in an unusual fashion. This will be described in more detail later. The inability to reduce residuals at specific nodes limits confidence in solutions in the region surrounding these nodes, as well as in the solution as a whole.

Other barriers prevent widespread use of CHAD in the nuclear industry. One such limitation is the lack of sophisticated mesh generation programs that are capable of meshing geometries that are exportable into a CHAD readable format. Another limitation is the lack of any graphical user interface (GUI) to facilitate interaction with the code. This lack of a GUI presents bottlenecks from model development through solution analysis. There exist many other areas in which CHAD may be improved, some of which are addressed over the course of this dissertation, and many which are not. Finally, there is almost no available documentation regarding CHAD. There is essentially no user's manual, only a draft form of a physics manual. This lack of documentation poses a significant hurdle to those newly acquainted with the code, and those attempting any detailed modifications within the code itself.

Limited literature exists in the public domain about CHAD. Several papers discuss the numerical schemes used in CHAD (O'Rourke, 1998 and O'Rourke, undated [a]), while others discuss applications (Thode, 1999 and O'Rourke, 1999) of code. At least one paper attempts to apply the numerical methodology in CHAD to Large-Eddy

Simulations (Haworth, 1996). A discussion of the latest paradigm in high-performance scientific computing uses CHAD as an example software code in a discussion on the infrastructure of large-scale software codes (Armstrong, 1999). Improvements in the performance of CHAD are discussed briefly in a paper from Argonne National Laboratory (Canfield, 2001a). Several other papers also briefly mention CHAD. These papers generally contained short discussions about CHAD in relation to other codes. It is assumed that there exist other papers within the classified domain pertaining to defense related applications.

1.3 Objective

The primary goal of this research work is to address limitations posed by the continuity solver thereby improving the ability of CHAD to efficiently solve reactor thermal hydraulics problems in the nuclear industry. Specifically, the goal is to replace the continuity equation solver through the addition of a new solver package in the hopes of improving the overall code execution speed.

A second major goal is to demonstrate CHAD's applicability to nuclear engineering problems by solving a large scale or computationally intensive problem relevant to the nuclear industry using the modifications to the continuity equation. Part of this goal includes the addition of several capabilities to CHAD. These new features will improve the functionality and the interactiveness of CHAD, including but not limited to greater program control options, improved solution monitoring, and the addition of a

buoyancy model and an incompressible equation of state. Some of these additional features will be utilized in solving the large scale nuclear-specific CFD calculation.

A third, and minor, goal of this work is to supplement the limited documentation available regarding CHAD in the form of a User's Manual. In writing this manual with new users of CHAD in mind, topics not adequately described in existing documentation will receive special attention. In particular, this manual will: 1) discuss technical issues surrounding installation and the compilation of CHAD on a new system; 2) provide an overview of the code's structure with the intent of providing a 'quick start' in locating and modifying CHAD; and, 3) document the lessons learned over the course of this dissertation so that new user's of CHAD have a head start on the learning curve and that the experience gained while working with the code is not lost.

This dissertation is organized as follows:

Chapter 2 discusses the three major phases of a CFD computation—preprocessing, solution, and postprocessing. As part of the solution phase, the relevant numerical schemes used by CHAD including its advection scheme and linearized solution algorithm are reviewed.

In Chapter 3 we take a closer look at the CHAD continuity equation. Here, we discuss how the continuity equation differs from its documented form, and we discuss changes to the continuity equation that were introduced in a previous work (Tzanos, 1998

[b]) and are reintroduced into a newer version of CHAD in this dissertation. We continue by discussing how the continuity equation is solved and how convergence is determined.

In Chapter 4 the review of the continuity equation continues, and it is shown how its terms may be constructed into a system of equations that can be solved with a matrix based solver.

In Chapter 5 the *Portable Extensible Toolkit for Scientific Computing* (PETSC) is introduced, and we show how it is used to solve the system of equations for the conservation of mass that were developed in Chapter 4. PETSc's implementation into CHAD is discussed.

In Chapter 6 the results to several benchmark problems are presented. Problems are solved to test the PETSc formulation developed in the preceding chapters. Several studies on the optimization of control parameters and how they affect the performance of CHAD are performed. Finally, several tools which were developed to assist in judging the convergence of CHAD are discussed.

Chapter 7 discusses the addition of an incompressible equation of state to CHAD. Results obtained using the new equation of state are also presented. Next, a capability is added to CHAD to simulate buoyant flow, and applied to a benchmark problem. Results of CHAD simulations to a buoyant flow benchmark problem are presented and discussed.

In Chapter 8, the incompressible equation of state, buoyancy model and PETSc are then used in an attempt to solve a computationally intensive simulation of a selected region within a reactor. Comparisons of this simulation are made against a similar calculation performed using the industry leading commercial CFD code Fluent. Strengths, weaknesses, and observations about CHAD's ability to solve such a model are discussed.

The dissertation finishes in Chapter 9 with a discussion of results presented in this work. Conclusions are drawn and areas in which future work is needed are highlighted.

An appendix to this dissertation stands alone as a Quick Start User's Guide for CHAD. It covers lessons learned in using CHAD and supplements the existing documentation for CHAD. It covers in thorough detail installation issues that one may encounter when installing CHAD and its supporting software. Additionally, the setup of models to be run by CHAD is discussed in detail covering input parameters, mesh setup, and boundary conditions. It also explains some topics in CHAD that are important for new users to understand, but are not covered elsewhere.

A separate appendix discusses the development of a mesh translation program that was developed for use on Microsoft Windows[®] to convert meshes developed using the commercial mesh generation program Gambit into a form usable by CHAD.

Chapter 2: The C.H.A.D. Code

2.1 C.H.A.D. (Computational Hydrodynamics for Advanced Design)

Originally developed under the Super Computing Automotive Applications Partnership in conjunction with the United States Council for Automotive Research and five Department of Energy Laboratories, CHAD is a recent code in a series of codes authored by the T-3 group at Los Alamos National Laboratory. Because of the emphasis for automotive applications, CHAD is often used as an acronym for the alternate program name “Computational Hydrodynamics for *Automotive* Design.” CHAD is the evolutionary progression of the KIVA code (Amsden, 1989).

CHAD is based on the finite volume method for solving the Navier-Stokes equations, which is typical of most modern CFD codes. Because CHAD is a finite volume based code, conserved variables are averaged over the volume of a cell associated with a node, with principal variables located at the nodes (which lie at the center) of each cell. CHAD utilizes a collocated, moving mesh that allows for the tracking of interfaces and shocks. The moving mesh scheme is particularly well adapted to the finite volume method (Hoffman, 1995). CHAD also includes models for chemical combustion and material stress.

CHAD is written to be highly portable on parallel platforms, with its platform dependent coding centrally located to remove it from the sections of the code that deal with the hydrodynamic calculations. The parallel communication in CHAD is performed

using a variation of the Message Passing Interface (MPI) called MPICH (Gropp, 1996a), an ANL distributed version of MPI (Gropp, 1999b). Operations performed on global data sets, such as gather-scatter routines to collect and distribute data between computer nodes, use a special performance oriented communication library. In particular, CHAD was written to utilize PGSLib (CPCA, 1997) and CommLib (Canfield, 2003b). While MPICH provides many of these parallel gather and scatter functions, the performance libraries purportedly provide superior message passing performance.

CHAD is written completely in FORTRAN 90 (F90) using many features not found in FORTRAN 77. One primary feature of F90 used extensively in CHAD is array operations. This feature allows operations to be performed on entire arrays with only one line of code. Other F90 features in CHAD include the use of module level files, interface blocks, and derived types.

Four versions of CHAD are discussed in this dissertation. Two of the latest versions of CHAD (referred to here as CHAD 4.0 and CHAD 5.0, though the actual release number differs slightly) were supplied by Los Alamos National Laboratory and contain many advancements and bug fixes not found in previous versions. The work discussed in this dissertation began with CHAD 4.0 and migrated to CHAD 5.0 upon its availability. Hereafter, references to CHAD 5.0 may be assumed to include all work originally implemented into derivative works originating from CHAD 4.0.

A third version of CHAD was provided by Argonne National Laboratory and is referred to as CHAD_ANL. CHAD_ANL is essentially CHAD 3.0 which was modified at Argonne in code extension work performed there. As it took a different evolutionary path, it contains features not found in CHAD 5.0. The copy of CHAD_ANL obtained for this work is one of several versions of CHAD 3.0 (with minor variations) that existed at ANL.

Early work at ANL was carried out on CHAD 3.0. One version of CHAD_ANL was inherited by the University of Illinois at Urbana-Champaign for further modification). This version included a mostly complete reformulated continuity equation, some trial capabilities for the level-set method, and portions of a trial two-phase homogeneous equilibrium model. However, at the time this dissertation began, a new mainstream version of CHAD had been released (CHAD 4.0) and shortly thereafter CHAD 5.0 was released. Since CHAD_ANL and CHAD 5.0 each contained worthwhile improvements, it was decided to merge those features in CHAD_ANL that are relevant to this dissertation into CHAD 5.0. To arrive at a “clean and up to date” version of CHAD, all features related to the modified continuity equation in CHAD_ANL were transferred over to CHAD 5.0. Parts of CHAD_ANL that dealt with the homogeneous equilibrium model for a two phase flow capability and those added to provide a level set front tracking capability were not transferred to the new version. Because of several substantial structural changes internal to CHAD that occurred between CHAD 3.0 and CHAD 5.0, this task required substantially more work than a simple “cut & paste” operation.

Within this dissertation, a new, combined version of CHAD 4.0, CHAD 5.0, and select portions of CHAD_ANL will be referred to as CHAD_UI. Unless otherwise specified, any changes made to CHAD as described in this dissertation are assumed to occur within CHAD_UI.

2.2 CHAD and the Three Steps of CFD Analyses

The typical process for performing a CFD analysis is followed when using CHAD—preprocessing, solution, and postprocessing. Preprocessing consists of model development and mesh generation, material specification, and setting boundary conditions. The solution phase involves solving governing equations that model the physical behavior of a system. Extraction and post processing of data to determine quantities of interest such as pressure drops and temperature changes across regions of interest, and presentation of results in an easily visualized form that can be utilized to draw conclusions from the simulation are performed in the postprocessing phase. In practice, additional tasks are performed than the three listed above, including problem identification, planning, and verification. Moreover, all of these steps are often part of a larger iteration process that ensures a CFD simulation truly represents both a physically meaningful solution, as well as the solution to the problem being solved.

We begin by discussing the preprocessing, solution and postprocessing stages of CFD analysis with emphasis on CHAD as opposed to a general CFD code. However, the discussion is tailored as an introduction for those readers unfamiliar with CFD analyses

and CFD software. Some particular details as they pertain to the research discussed in this dissertation will be noted.

2.2.1 Preprocessing Phase: Generating a CHAD Mesh

CHAD uses a node-centered finite volume scheme extended to unstructured meshes (Selmin, 1992). Unstructured meshes are useful in modeling complex geometries. In a structured mesh, nodes follow a regular pattern and therefore the identity of adjacent nodes is predictable. However, regular patterns are not convenient for meshing irregular shapes. In unstructured meshes, node patterns are not a constraint and therefore irregular shapes are easily meshed. However, additional data structures are required to keep track of adjacent nodes, and additional numerical difficulties may result from the use of certain cell shapes typically found in unstructured meshes.

The control volume associated with a node for CHAD's integral equations is the union of all *sub-volumes* that touch a node (Figures 2.1 and 2.2.) A sub-volume is the region of space surrounding a node that is associated with that node and is bounded by the mesh grid and a "median mesh face." The outside faces formed by these sub-volumes compose the median mesh face, of which the face midpoints are the centroids of the faces and the element midpoints are the centroids of the elements. In CHAD terminology, the node of a control volume is designated by v . A node at the opposite end of a connection is designated by $v\alpha$. Median mesh quantities are designated using the subscript α . No explicit subscript is used to describe CHAD connections.

CHAD is written using edge-based data structures. As a result, information regarding node neighbors is not stored *directly* in the “computational grid” part of the code. Instead, a two-dimensional array is used to track the relationship between any two nodes that are adjacent to each other. One dimension of the array is used to keep track of mesh grid edges (connections) that connect two nodes, while the second array dimension contains the identification numbers of the nodes at each end of a connection. These connections are part of the “mesh grid,” but they are not part of the “computational grid” used by CHAD. They are primarily used as a computational tool for transferring information between node neighbors. When information is needed regarding neighbors of a particular node, it is obtained through calls to the communication routines which transfer information between nodes and edges. See Appendix A for additional discussion of the communication routines.

Mesh generation for CHAD follows a somewhat standard approach that is used in the generation of most CFD meshes. First, a computer aided design program (CAD) such as ProEngineer (ProE, 2005) is used to create a set of surfaces or frames. This framework is then exported in a general format which can then be imported by a variety of meshing programs, such as CUBIT (Sandia National Lab, 2000).

Within a mesh generation program, the imported framework is first “sealed off” to close numerical gaps and open faces, creating a three dimensional body. A grid is then overlaid on this body and a three dimensional mesh is created. This mesh is used to

create the nodes and control volumes which are used to develop the set of discrete equations from the set of governing partial differential equations.

Various cell shapes and sizes may be chosen by the user during this process. Boundary conditions are then assigned to nodes, surfaces and volumes as desired. Because of the way boundary conditions are implemented within CHAD, only simple boundary conditions are directly available for use by the CHAD executable program. Though boundary conditions are read into CHAD through the mesh file, the specifics regarding those boundary conditions must be explicitly programmed into CHAD. For example, in a simulation involving a single, moving wall, it is simple to provide boundary conditions and specify those conditions through the input file. However, in the case of multiple walls moving with different velocities, custom FORTRAN must be built into the CHAD executable. The same holds true for various other boundary conditions typically found in CFD calculations, such as different inflow velocities, wall temperatures or heat transfer rates, or source terms.

For typical hydrodynamic calculations such as shock wave propagation in CHAD, such versatility is not so much of a requirement, as the evolution of the simulation typically can progress to solution through the specification of the initial state of the system (for example in a shock problem, one region of the problem can be initialized at a higher pressure than the rest of the system.) For CFD calculations this is generally not true and therefore the process of setting up a problem involves both customizing a mesh as well as customizing the corresponding executable.

Most commercial mesh generator programs are capable of exporting their meshed geometries into a variety of common CFD formats. These common formats then require a translation program in order to be loaded into a CFD code. Some mesh generation programs provide specific formats that may be directly loaded into a CFD code without the translation step.

CHAD uses its own mesh file format and therefore is not directly compatible with standard mesh file formats. Additionally, the package distributed with CHAD does not provide any input translators for converting generic mesh formats into a form usable with CHAD. In this dissertation, simple CHAD meshes were constructed using a FORTRAN routine, while more complicated meshes were developed using the commercial mesh generation utility GAMBIT (Fluent Inc, 2005), and a custom mesh translator, described in Appendix B, was developed especially for this dissertation and for general use. Briefly, this mesh translation utility generates the structure of the mesh in a CHAD readable format, but does not generate boundary condition sets required for the problem. As CHAD boundary conditions often require custom programming, executables implementing specified boundary conditions were developed for individual applications and typically did not utilize mesh information.

2.2.2 Solution Phase: The CHAD Solver

CHAD is based on an explicit/implicit numerical method to simulate fluid flow on unstructured meshes (O'Rourke, 1998a). CHAD solves integral equations for mass,

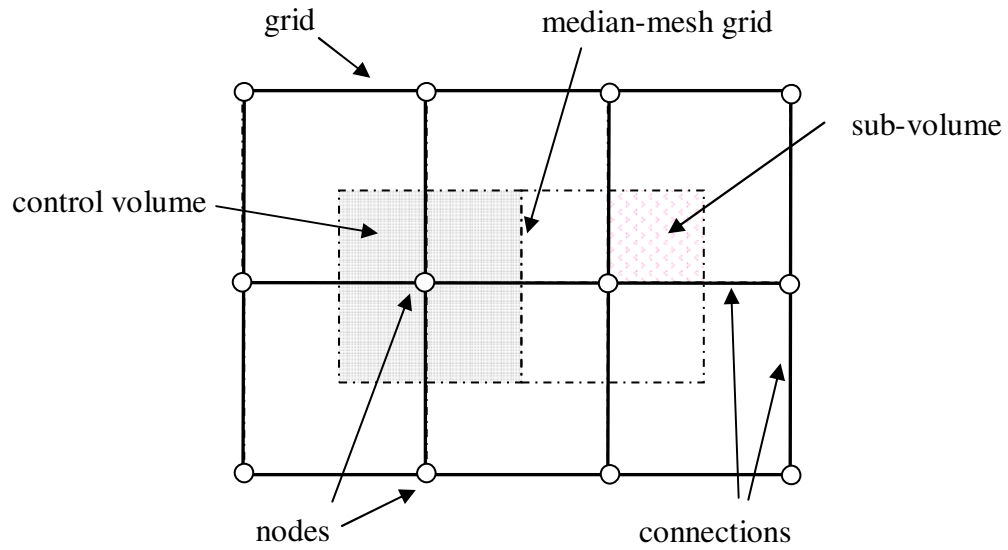


Figure 2.1: An example of a structured grid and median mesh highlighting a finite volume and a sub-volume.

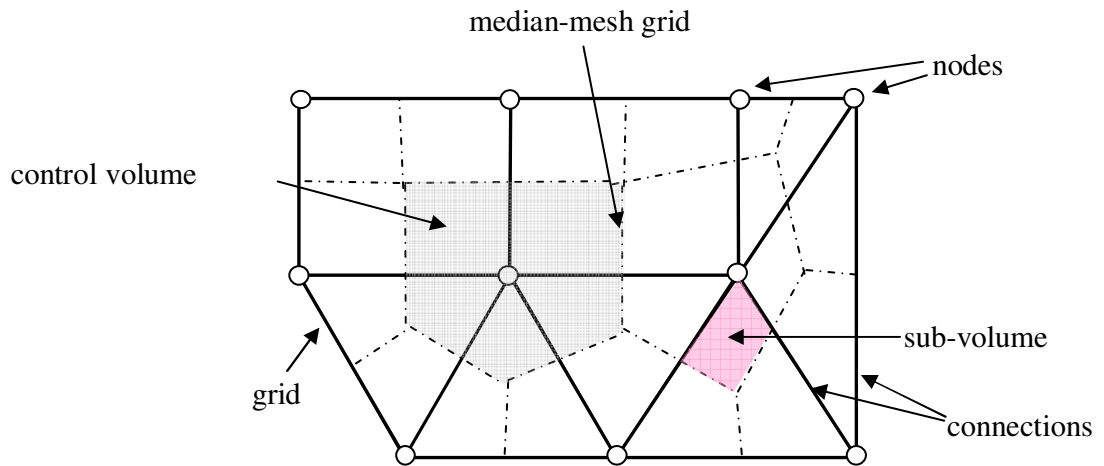


Figure 2.2: An example of an unstructured grid showing a finite volume and a sub-volume.

momentum, enthalpy, and mixture. The computational phase is divided into three stages. Mixture, mass, momentum and enthalpy equations are solved in the first stage. In the second stage, the species mass fraction equations are solved. Turbulence equations are solved in the third stage, if the flow is turbulent.

Integrating the time-dependent Navier-Stokes and energy equations over a control volume yields

$$\frac{d}{dt} \iiint_V \rho dV + \iint_S \rho (\mathbf{u} - \mathbf{v}) \cdot \mathbf{n} dA = 0 \quad (2.1)$$

$$\frac{d}{dt} \iiint_V \rho \mathbf{u} dV + \iint_S \left[\rho \mathbf{u} (\mathbf{u} - \mathbf{v}) \cdot \mathbf{n} + \left(\frac{p}{a^2} + \sigma_T \frac{2}{3} \rho K \right) \mathbf{n} \right] dA = \iint_S (\tau_L - \sigma_T \tau_T) \cdot \mathbf{n} dA \quad (2.2)$$

$$\begin{aligned} \frac{d}{dt} \iiint_V \rho h dV + \iint_S \rho h (\mathbf{u} - \mathbf{v}) \cdot \mathbf{n} dA &= \iiint_V \left(\frac{\partial p}{\partial t} + \mathbf{u} \cdot \nabla \frac{p}{a^2} \right) dV - \iint_S (Q_L - \sigma_T Q_T) \cdot \mathbf{n} dA \\ &+ \iiint_V (\tau_L \cdot \nabla \mathbf{u} + \sigma_T \rho \varepsilon) dV \end{aligned} \quad (2.3)$$

These three equations are discretized and solved implicitly from time t^n to t^{n+1} . The discretized momentum equation is given by

$$\begin{aligned} \frac{\rho_v^{n+1} \mathbf{u}_v^{n+1} V_v^{n+1} - \rho_v^n \mathbf{u}_v^n V_v^n}{\Delta t} &+ \sum_\alpha \rho_\alpha \mathbf{u}_\alpha [\tilde{\mathbf{u}}_\alpha \cdot \mathbf{A}_\alpha^{n+1}] + \sigma_T \sum_\alpha \left[\left(\frac{2}{3} \rho K \right)_\alpha^n - \left(\frac{2}{3} \rho K \right)_v^n \right] \mathbf{A}_\alpha^{n+1} \\ &+ \sum_\alpha \left[\left(\frac{2}{3} \rho K \right)_\alpha^n - \left(\frac{2}{3} \rho K \right)_v^n \right] \mathbf{A}_\alpha + \frac{1}{a^2} \sum_\alpha [p_\alpha - p_v^{n+1}] \mathbf{A}_\alpha^{n+1} \\ &= \sum_\alpha (\tau_\alpha - f_v \tau_v) \cdot \mathbf{A}_\alpha^{n+1} + \mathbf{N}_v + (\mathbf{S}_u)_v \end{aligned} \quad (2.4)$$

where it should be noted that there are two separate cell-face velocity approximations

$$\begin{aligned} \tilde{\mathbf{u}}_\alpha = & \frac{\mathbf{u}_v^{n+1} - \mathbf{u}_{v\alpha}^{n+1}}{2} - \frac{\tilde{\Delta t}_\alpha}{\rho_\alpha^n} \left\{ \frac{1}{a^2} (\nabla p)_\alpha^{n+1} + \sigma_T \left(\nabla \frac{2}{3} pK \right)_v^n \right. \\ & \left. - \frac{1}{2} \left[\frac{1}{a^2} (\nabla p)_v^{n+1} + \sigma_T \left(\nabla \frac{2}{3} pK \right)_v^n - \frac{1}{a^2} (\nabla p)_{v\alpha}^{n+1} + \sigma_T \left(\nabla \frac{2}{3} pK \right)_{v\alpha}^n \right] \right\} \end{aligned} \quad (2.5)$$

and

$$\left. \frac{D\mathbf{u}}{Dt} \right|_\alpha = \frac{1}{\rho_\alpha^n} \left[-\frac{1}{a^2} (\nabla p)_\alpha^{n+1} - \sigma_T \left(\nabla \frac{2}{3} \rho K \right)_\alpha^n + \frac{1}{2} (\nabla \cdot \boldsymbol{\tau}|_v + \nabla \cdot \boldsymbol{\tau}|_{v\alpha}) \right]. \quad (2.6)$$

The computation of the cell-face tilde velocity in equation (2.5) uses a modified form of the Rhie-Chow method (Rhie, 1983). The formulation in equation (2.5) differs from the Rhie-Chow method in that the gradient calculated at the cell-faces is corrected to keep the component of the gradient between two nodes consistent with the difference between the values of the two nodes (O'Rourke, 1989c).

The use of two separate equations to approximate cell-face velocities is part of an effort to eliminate alternate node decoupling using a fourth order node coupler (equation 2.5) while retaining stability using upwinding of the advection term (equation 2.6). A generalization of the advection scheme is presented below.

2.2.2.1 NO-UTOPIA

CHAD utilizes an advanced advection scheme based upon an implicit-explicit method (Collins et al., 1995) that was generalized for hybrid unstructured grids (O'Rourke and Sahota, 1997). In this scheme, upwinding is in the direction of the material velocity. The essence of this scheme is not directly relevant to the work performed in this dissertation. However, any modification introduced in the CHAD code

necessitates an understanding of this advection scheme. Hence, a brief description is given below (Sahota, 1997). To review this scheme, we begin with the one dimensional, linear advection equation of the form

$$\frac{\partial q}{\partial t} = u_0 \frac{\partial q}{\partial x} = S(x, q). \quad (2.7)$$

Equation (2.7) in its finite difference form can be written as

$$\frac{q_i^{n+1} - q_i^n}{\Delta t} + u_0 \frac{q_{i+1/2} - q_{i-1/2}}{\Delta x} = S(q^{n+1}). \quad (2.8)$$

Discrete variables at the boundary of a cell are related to nodal variables via the material time derivative

$$\frac{Dq}{Dt} = S(x, q). \quad (2.9)$$

The value of the boundary quantities $q_{i+1/2}$ are approximated at an advanced time $t^n + \phi \Delta t$, where the value of ϕ is between $1/2$ and 1 for stability and is determined by the Courant Number C_0 . When $C_0 \leq 1/2\phi$

$$\left. \frac{Dq}{Dt} \right|_{i+1/2} = \frac{q_{i+1/2} - \left[q_i^n + \left(\frac{\partial q}{\partial x} \right)_{l,i}^n \Delta x \left(\frac{1}{2} - \phi C_0 \right) \right]}{\phi \Delta t} \quad (2.10)$$

where the l subscript indicates the use of a *limited gradient* which is a generalization of a MUSCL limiting (Van Leer, 1979). When $C_0 \geq 1/2\phi$

$$\left. \frac{Dq}{Dt} \right|_{i+1/2} = \frac{q_{i+1/2} - \left[\left(\phi - \frac{1}{2C_0} \right) q_i^{n+1} + \left(1 - \phi - \frac{1}{2C_0} \right) q_i^n \right]}{\frac{\Delta t}{2C_0}} \quad (2.11)$$

where the CCG (Collins, Collela and Glaz, 1995) scheme provides a method for choosing ϕ as

$$\phi = \begin{cases} \frac{1}{2}, & \text{if } C_0 \leq 1 \\ 1 - \frac{1}{2C_0}, & \text{if } C_0 > 1 \end{cases}. \quad (2.12)$$

Equation (2.9) is discretized as

$$\left. \frac{Dq}{Dt} \right|_{i+\frac{1}{2}} = \frac{1}{2} \left[S_i(q^{n+1}) + S_{i+1}(q^{n+1}) \right] \quad (2.13)$$

Complete forms of these equations may be found in the CHAD physics manual (O'Rourke-b, undated) and in a paper discussing the generalization of the CCG scheme (O'Rourke, 1997a).

The above discretization results in the following desirable properties in the numerical scheme:

- conservative
- second order accurate in unsteady regions
- second order accurate in space in steady regions
- max norm diminishing for all time steps

2.2.2.2 The SIMPLE Algorithm

The solution algorithm is modeled after the SIMPLE method (Patankar, 1980 and Ferziger, 2002). In the SIMPLE method as implemented in CHAD, systems of coupled, nonlinear equations are solved by linearizing each equation and solving for the change in the associated variable while holding other variables constant—a process depicted in Figure 2.1. This is true for every equation except the pressure equation. In the pressure equation in CHAD, velocities are permitted to change during the iteration process. These linearized equations are solved sequentially and repeatedly using CHAD's default solver, a matrix-free form of the Generalized Conjugate Residual solver (GCR). The solver does not require knowledge of matrix coefficients, only the residuals of the equations being solved. This matrix free solver was used because computing coefficients can be costly, and because mesh connectivity information is not readily available in CHAD (O'Rourke, b, undated).

The linearized momentum equations for Δu , Δv and Δw components of velocity are solved simultaneously by a vectorized form of GCR while holding pressure and enthalpy constant. In solving these equations, an estimated value of pressure is used, which is given by

$$p_{est} = p + \Delta t \frac{p - p_{old}}{\Delta t_{old}}. \quad (2.14)$$

After solving the momentum equations, the changes in velocity are used to update the existing values for u , v , and w . Additionally, the cell-face velocities (u_α , v_α , and w_α)

and cell-face tilde velocities ($\tilde{u}_\alpha, \tilde{v}_\alpha$, and \tilde{w}_α) are updated. As the cell-face tilde velocities are updated, so too is the volumetric flow rate crossing the cell-face boundaries.

After the momentum equation is solved, the Courant number is recomputed. This process includes recomputing cell-face boundary density, cell-face velocity, and cell-face enthalpy.

The next step in CHAD's implementation of the SIMPLE algorithm is the solution of the energy equation for changes in nodal and cell-face specific enthalpy ($\Delta h, \Delta \tilde{h}_m$) and temperature ($\Delta T, \Delta \tilde{T}_m$). When solving this equation, pressure and velocity are held constant. After obtaining solutions using a scalar version of the GCR, changes in enthalpy and temperature are used to compute changes in nodal and cell-face density. These changes are then used to update specific enthalpy, temperature and density.

The last linearized equation is solved to compute pressure. However, unlike the momentum and enthalpy equations, the solution to the pressure equation allows velocity and enthalpy to *float* during the iteration process. Therefore, along with the changes in pressure, Δp , changes in nodal and cell-face velocity, density and enthalpy are also computed. Additionally, as a result of computing changes in velocity, the cell-face tilde velocities are also recomputed. Again, this necessitates the recomputation of the volumetric flow rate through median-mesh boundaries.

After obtaining a solution to the pressure equation, the set of linearized equations is checked for convergence. If the linearized equations do not satisfy predetermined convergence criteria, the solution procedure repeats itself by first computing an equation of state then starting the iteration procedure anew by resolving the momentum equations.

The number of iterations required for the continuity equation to achieve convergence tends to be greater than those of the other linearized equations. Thus, the solution of the continuity equation consumes a large fraction of the total CPU time.

CHAD was developed to model compressible flows. As such, closure to the system of equations is obtained using an equation of state. While the computation of the equation of state exists outside of the SIMPLE algorithm, it carries a noteworthy consequence. In completely incompressible flow problems in which there is no heat transfer involved, the solution to the enthalpy equation is still computed, adding additional, but unnecessary computations.

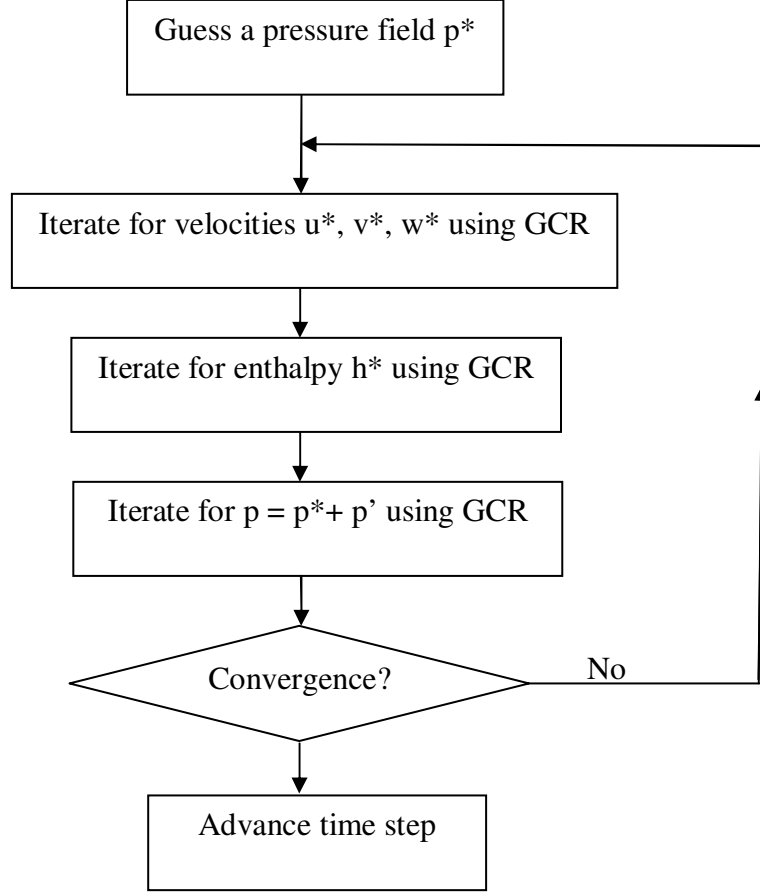


Figure 2.3: Simple-Like Algorithm used in CHAD

2.2.2.3 The Generic q Equation

The momentum, energy, species transport and turbulence equations all have the form

$$\begin{aligned}
 & \frac{\rho_v^{n+1} \mathbf{u}_v^{n+1} V_v^{n+1} - \rho_v^n \mathbf{u}_v^n V_v^n}{\Delta t} + \sum_{\alpha} \rho_{\alpha} q_{\alpha} \left[\tilde{\mathbf{u}}_{\alpha} \cdot \mathbf{A}_{\alpha}^{n+1} - \frac{\phi_{\alpha}}{\Delta t} \right] \\
 & = \sum_{\alpha} \left((D_q)_{\alpha} (\nabla q)_{\alpha}^{n+1} - f_v (D_q)_v (\nabla q)_v^{n+1} \right) \cdot \mathbf{A}_{\alpha}^{n+1} + (S_q)_v
 \end{aligned} \tag{2.15}$$

where q is a generic variable and D is the *diffusion coefficient* and S is the source term.

After some manipulation of equation (2.15) and the discretized continuity equation, a nonlinear residual for the q equation may be written as (O'Rourke and Sahota, 1997).

$$\begin{aligned} (R_q)_v = q_v^{n+1} - q_v^n + \frac{\Delta t}{\rho_v^{n+1} V_v^{n+1}} \left\{ \sum_{\alpha} \rho_{\alpha} (q_{\alpha} - q_v^n) \left[\tilde{\mathbf{u}}_{\alpha} \cdot \mathbf{A}_{\alpha}^{n+1} - \frac{\phi_{\alpha}}{\Delta t} \right] \right. \\ \left. \sum_{\alpha} \left[(D_q)_{\alpha} (\nabla q)_{\alpha}^{n+1} - f_v (D_q)_v (\nabla q)_v^{n+1} \right] \cdot \mathbf{A}_{\alpha}^{n+1} - (S_q)_v \right\} = 0 \end{aligned} \quad (2.16)$$

Linearizing equation (2.16) yields

$$\begin{aligned} (R_q)_{v,lin} = (R_q)_v + \delta q_v \frac{\Delta t}{\rho_v V_v^{n+1}} \left[\sum_{\alpha} \rho_{\alpha} \delta q_{\alpha} \left(\tilde{\mathbf{u}}_{\alpha} \cdot \mathbf{A}_{\alpha}^{n+1} - \frac{\phi}{\Delta t} \right) \right. \\ \left. + \sum_{\alpha} \delta \rho_{\alpha} (q_{\alpha} - q_v^n) \left(\tilde{\mathbf{u}}_{\alpha} \cdot \mathbf{A}_{\alpha}^{n+1} - \frac{\phi}{\Delta t} \right) + \sum_{\alpha} \rho_{\alpha} (q_{\alpha} - q_v^n) (\delta \tilde{\mathbf{u}}_{\alpha} \cdot \mathbf{A}_{\alpha}^{n+1}) \right. \\ \left. - \sum_{\alpha} \left[(D_q)_{\alpha} (\nabla \delta q)_v - f_v (D_q)_v (\nabla \delta q)_v \right] \cdot \mathbf{A}_{\alpha}^{n+1} - \delta (S_q)_v \right. \\ \left. - \frac{\delta \rho_v}{\rho_v} (R_{q,v} - q_v + q_v^n) \right] \end{aligned} \quad (2.17)$$

which is the linearized form of the generic equation that is solved within the SIMPLE algorithm. Since implementing a reformulated form of the continuity equation is one of the objectives of this dissertation, a discussion of the continuity equation is postponed until Chapter 3, which is devoted entirely to the continuity equation.

2.2.2.4 Restart Capability in CHAD

CHAD provides a restart capability that allows calculations to be initialized from previous calculations. During its solution phase, CHAD writes dump file containing information about the state of the system. Not all variables are written to the dump file; only those that are the most fundamental. Other data are calculated at the time of a restart as required. Dump files are written at intervals defined by both the problem time and a

cycle interval. When it is desired to resume a simulation from a previous calculation, a restart flag is specified in the input file and the appropriate dump file is referenced. Unlike some commercial CFD codes, CHAD requires that the same mesh be used on a restart calculation as was used to generate the restart dump file. There is no facility for interpolating a solution onto a new mesh grid.

2.2.3 Postprocessing: Visualizing CHAD Results

At the termination of the solution process in CHAD, either via satisfaction of the convergence criteria or having met some other limit such as maximum number of iterations, results are written in output files. Output files are typically written in formats suitable for viewing in a postprocessor. In CHAD, supported formats are Enight (CEI, 2005) and Generalized Mesh Viewer (Ortega, 2004). In this work Los Alamos National Laboratory's GMV was used for postprocessing. With GMV, or essentially any other postprocessor, data may be visualized with various *plot types* including *vector*, *contour* and *isosurfaces*. Additionally, data may be plotted along lines and over selected planes. These visualization utilities allow a CFD user to interpret simulation results and apply them for design, optimization or analysis.

CHAD provides the ability to control the frequency of the graphics output based on problem time or a number of increments solution cycle counter. Additionally, through user FORTRAN interfaces, additional information may be output into the graphics file, allowing the user to view code calculated data or data not normally written out in output files.

Chapter 3: Revisions to the CHAD Continuity Solver

3.1 The CHAD Continuity Solver

In an effort to modify the CHAD code to make it more suitable for use in the nuclear power industry, several modifications and improvements can be made. Since CHAD is a slow code, one desired change is to make it run faster. This would most likely require changes in its continuity solver since it is suspected to be the bottle neck in CHAD's speed. Another factor restricting CHAD's use in the nuclear industry is that the continuity equation, in its original form, is subject to convergence difficulties when a flow field is nearly incompressible (Tzanos, undated [a]). A continuity equation that is capable of solving incompressible flow fields, typically used to model pressurized water reactors, would constitute a significant improvement in CHAD's capabilities.

A third desirable feature in CHAD would be the availability of several solvers for the continuity equation. For example, it might be desirable to select a solver that performs better in parallel computations than in serial computations, such as what might occur while conducting a large simulation. CHAD's current solver, GCR, is by default the only solver available for solving the continuity equation (two other solvers exist but there is no interface or code from which to access them). Since no single solver algorithm is optimal for all situations, multiple solvers that are selectable at run time would also improve CHAD.

3.2 The CHAD Continuity Equation

In the Navier-Stokes equations, velocities are obtained by solving momentum equations. Similarly, enthalpy and temperature are obtained by solving energy equations. This leaves pressure to be obtained using the continuity equation. Unfortunately, the continuity equation does not contain pressure terms. To overcome this problem, the continuity equation is typically combined with the momentum equation to form an equation for pressure. CHAD uses this modified form of the continuity equation to solve for pressure (Ferziger, 2002).

The integral form of the CHAD continuity equation is given by

$$\frac{d}{dt} \iiint_V \rho dV + \iint_S \rho (\mathbf{u} - \mathbf{v}) \cdot \mathbf{n} dA = 0 \quad (3.1)$$

while the integral form of the momentum equation is given by

$$\frac{d}{dt} \iiint_V \rho \mathbf{u} dV + \iint_S \left[\rho \mathbf{u} (\mathbf{u} - \mathbf{v}) \cdot \mathbf{n} + \left(\frac{p}{a^2} + \sigma_T \frac{2}{3} \rho K \right) \mathbf{n} \right] dA = \iint_S (\tau_L - \sigma_T \tau_T) \cdot \mathbf{n} dA. \quad (3.2)$$

Discretization of the continuity equation gives

$$\frac{V}{\partial t} (\rho - \rho^n) + \sum_{\alpha} \rho_{\alpha} \tilde{\mathbf{u}}_{\alpha} \cdot \mathbf{A}_{\alpha} = 0. \quad (3.3)$$

In CHAD, the cell-face velocity $\tilde{\mathbf{u}}_{\alpha}$ is given by

$$\begin{aligned} \tilde{\mathbf{u}}_{\alpha} = & \frac{\mathbf{u}_v^{n+1} - \mathbf{u}_{v\alpha}^{n+1}}{2} - \frac{\tilde{\Delta t}_{\alpha}}{\rho_{\alpha}^n} \left\{ \frac{1}{a^2} (\nabla p)_{\alpha}^{n+1} + \sigma_T \left(\nabla \frac{2}{3} p K \right)_v^n \right. \\ & \left. - \frac{1}{2} \left[\frac{1}{a^2} (\nabla p)_v^{n+1} + \sigma_T \left(\nabla \frac{2}{3} p K \right)_v^n - \frac{1}{a^2} (\nabla p)_{v\alpha}^{n+1} + \sigma_T \left(\nabla \frac{2}{3} p K \right)_{v\alpha}^n \right] \right\}. \end{aligned} \quad (3.4)$$

Substituting equation (3.4) into equation (3.3) yields an equation that is solved for pressure. The term a^2 is a pressure gradient scaling term (PGS) which is introduced to increase the calculation efficiency of compressible flows that have a nearly uniform pressure field (Ramshaw, 1984).

Equations (3.3) and (3.4) are used within CHAD's implementation of the SIMPLE algorithm to calculate pressure, or more precisely a change in pressure that is added to the current nodal pressure field to update it so that the updated pressure field is consistent with the velocity field and satisfies the continuity equation. The solution strategy of CHAD's continuity solver is to compute a change in nodal pressure which is a minimized linear residual of a residual form of the combined momentum-pressure equation.

The residual of the continuity equation is given by

$$(R_p)_v = \rho_v - \rho_v^n \frac{V_v^n}{V_v^{n+1}} + \frac{\Delta t}{V_v^{n+1}} \left[\sum_{\alpha} \rho_{\alpha} (\tilde{\mathbf{u}}_{\alpha} \cdot \mathbf{A}_{\alpha}^{n+1}) \right] \quad (3.5)$$

while the linearized residual is given by

$$(R_p)_{v,lin} = (R_p)_v + \delta \rho_v + \frac{\Delta t}{V_v^{n+1}} \left[\sum_{\alpha} \rho_{\alpha} \delta \tilde{\mathbf{u}}_{\alpha} \cdot \mathbf{A}_{\alpha}^{n+1} + \sum_{\alpha} \delta \rho_{\alpha} \left(\tilde{\mathbf{u}}_{\alpha} \cdot \mathbf{A}_{\alpha}^{n+1} - \frac{\phi_{\alpha}}{\Delta t} \right) \right] \quad (3.6)$$

where

$$\delta \rho_v = \left. \frac{\partial \rho}{\partial p} \right|_{T,v}^n \delta p_v + \left. \frac{\partial \rho}{\partial T} \right|_{p,v}^n \delta T_v \quad (3.7)$$

$$\delta \tilde{\mathbf{u}}_{\alpha} = \frac{\delta \mathbf{u}_v + \delta \mathbf{u}_{v\alpha}}{2} - \frac{\tilde{\Delta t}}{a^2 \rho_{\alpha}^n} \left\{ (\nabla \delta p)_{\alpha} - \frac{1}{2} [(\nabla \delta p)_v + (\nabla \delta p)_{v\alpha}] \right\} \quad (3.8)$$

$$\delta\rho_\alpha = \delta\rho_{\alpha,old} - \frac{\Delta t}{2\rho_\alpha^n} \left\{ \frac{1}{V_\nu^{n+1}} \sum_\beta (\delta\tilde{\mathbf{u}}_\beta - \delta\mathbf{u}_\nu) \cdot \mathbf{A}_\beta^{n+1} + \frac{1}{V_{\nu\alpha}^{n+1}} \sum_\beta (\delta\tilde{\mathbf{u}}_\beta - \delta\mathbf{u}_{\nu\alpha}) \cdot \mathbf{A}_\beta^{n+1} \right\} \quad (3.9)$$

$$\delta\rho_{\alpha,old} = \begin{cases} \left(1 - \frac{1}{C_\alpha}\right) \delta q_\nu & \text{if } C_\alpha > 1 \text{ and } \mathbf{u}_{rel} \cdot \mathbf{A}_\alpha^{n+1} > 0 \\ \left(1 - \frac{1}{C_\alpha}\right) \delta q_{\nu_\alpha} & \text{if } C_\alpha > 1 \text{ and } \mathbf{u}_{rel} \cdot \mathbf{A}_\alpha^{n+1} < 0 \\ 0 & \text{otherwise} \end{cases} \quad (3.10)$$

and β is an index that denotes computational boundary faces.

An approximation for $\delta\mathbf{u}_\nu$ is obtained by ignoring changes in convective and viscous terms, and by ignoring changes in nodal densities. Then, $\delta\mathbf{u}_\nu$ is given by

$$\delta\mathbf{u}_\nu = \frac{\Delta t (\nabla \delta p)_\nu}{\rho_\nu a^2}. \quad (3.11)$$

Nodal temperatures are obtained from the linearized temperature equation, ignoring changes in convection and heat conduction terms as well as nodal density changes

$$\delta T_\nu = \frac{\delta p_\nu \left(\frac{1}{\rho_\nu} - \frac{\partial h}{\partial p} \Big|_{T,\nu}^n \right) + \delta\mathbf{u}_\nu \cdot \left[\frac{\Delta t}{a^2 p} (\nabla p)_\nu - \mathbf{u}_\nu + \mathbf{v}_\nu \right]}{c_{p,\nu}^n} \quad (3.12)$$

where \mathbf{v}_ν is the control volume velocity. Equations (3.6)-(3.12) are solved for pressure implicitly within the pressure iteration (O'Rourke, undated [a]).

The procedure described above is based on the CHAD manual (O'Rourke, undated [a]). However, several of the equations found in the source code of CHAD deviate from those given above and are discussed later in this chapter.

3.3 The Reformulated Continuity Equation

In an effort to overcome the problems associated with the poor performance of the continuity solver, a reformulated continuity equation was introduced in CHAD at Argonne National Lab (ANL) using a different cell-face mass flux approximation (Peric, 1998). The reformulation was carried out in such a way as to allow the construction of a diagonally dominant matrix system for the pressure equation that may be solved using a generic black box matrix solver like the PETSc toolkit. Specifically, the continuity equation was derived such that a system of equations of the form $\underline{\underline{A}}\underline{x} = \underline{b}$ could be developed and numerically solved. We describe the reformulation of the continuity equation used in CHAD_ANL and its reinsertion into CHAD 5.0 here, and we describe the development of the system of equations and their implementation into the PETSc Toolkit in Chapter 4. This reformulation carried out at ANL distinguishes it from the LANL versions of CHAD, versions 4.0 and 5.0.

The reformulation carried out at ANL addresses several of the shortcomings associated with CHAD's continuity equation. The formulation of a system of equations in matrix forms allows for the addition of new solvers using the PETSc Toolkit. Additionally, diagonally dominant systems of equations are more easily solved for incompressible flows.

The derivative of the residual of the continuity equation with respect to pressure is

$$\frac{\partial R_p}{\partial p} \approx \frac{\partial \rho}{\partial p} + \frac{\partial R_p}{\partial \tilde{\mathbf{u}}_\alpha} \frac{\partial \tilde{\mathbf{u}}_\alpha}{\partial p} + \frac{\partial R_p}{\partial \rho_\alpha} \frac{\partial \rho_\alpha}{\partial p} \quad (3.13)$$

In CHAD_ANL, the derivative of this equation is approximated by the first term on the right hand side

$$\frac{\partial R_p}{\partial p} \approx \frac{\partial \rho}{\partial p} \quad (3.14)$$

In incompressible flows, as will be the case for most liquid water based applications in nuclear engineering, such as in a pressurized water reactor, this residual term is small and the neglected terms become significant. The neglected terms are significant for a second reason. If the derivative of density with respect to pressure is small, corresponding coefficients in the matrix are also small, with weak diagonal dominance. As matrix solvers generally require diagonal dominance for good performance, it was useful to rewrite the continuity equation in a form that ties in the significance of the momentum terms to the residual of the continuity equation. Inclusion of these terms would provide diagonal dominance to a matrix for incompressible flow analysis.

Tzanos rewrote the continuity equation in CHAD using a first order upwind differencing scheme (Tzanos, 1998 [b]) that was intended to form the basis of a higher order scheme. In the reformulated scheme, the cell-face velocity is given by

$$\tilde{\mathbf{u}}_\alpha = \left[\frac{\sum_{nb} C_{nb} \mathbf{u}_{nb} + \frac{V}{dt} \rho^n \mathbf{u}^n + Q_{u,v}}{C_v} \right] - \frac{\bar{V}_v}{C_v} \bar{i}_i (\nabla p)_\alpha \quad (3.15)$$

where \mathbf{u}_{nb} is the velocity at the neighboring (nb) node $v\alpha$ across connection α and node v ,

$$C_{nb} = -\frac{1}{2} [1 - VFX(\cdot)] m_\alpha(\cdot) \quad (3.16)$$

$$C_v = \left\{ \frac{V_v}{\delta t} \rho_v + \frac{1}{2} [1 + VFX(:)] m_\alpha(:) \right\} \quad (3.17)$$

$$VFX = \begin{cases} +1, m_\alpha > 0 \\ -1, m_\alpha < 0 \end{cases} \quad (3.18)$$

$$\frac{\bar{V}_v}{C_v} = \frac{1}{2} \left[\frac{V_v}{C_v} + \frac{V_{v\alpha}}{C_{v\alpha}} \right] \quad (3.19)$$

$$(\nabla p)_\alpha = (\tilde{\nabla} p)_\alpha - \frac{(\tilde{\nabla} p)_\alpha \cdot \bar{\mathbf{d}}_\alpha}{|\bar{\mathbf{d}}_\alpha|^2} \bar{\mathbf{d}}_\alpha + \frac{p_{v\alpha} - p_v}{2|\bar{\mathbf{d}}_\alpha|^2} \bar{\mathbf{d}}_\alpha \quad (3.20)$$

$$(\tilde{\nabla} p)_\alpha = \frac{1}{2} [(\nabla p)_v + (\nabla p)_{v\alpha}] \quad (3.21)$$

The terms of equation (3.19), when multiplied by the pressure gradient, represent the contribution to the flow between cells due to differences in cell pressures. Thus, it is the terms of equation (3.19) which will be used to construct part of the coefficients of the matrix, which will be described shortly.

This reformulation is used to calculate the median-mesh (tilde) velocities $\tilde{\mathbf{u}}$ in several places within the code. The reformulated tilde velocities are calculated initially at the beginning of the outer iteration portion of the SIMPLE algorithm. Then, during the inner iterations of the momentum equation, changes in the nodal velocities require that the reformulated tilde velocities be recomputed. In CHAD, that change in tilde velocities is computed as

$$\tilde{\mathbf{u}} = \frac{1}{2} (\delta \mathbf{u}_v + \delta \mathbf{u}_{v\alpha}) \quad (3.22)$$

while in CHAD_ANL they are updated as

$$\tilde{\mathbf{u}}_\alpha = \left[\frac{\sum_\alpha C_{nb} \delta \mathbf{u}_{nb} + \frac{V}{dt} \rho^n \mathbf{u}^n + Q_{u,v}}{C_v} \right]. \quad (3.23)$$

During the iterative solution of the momentum equation, the change in tilde velocities along walls is set equal to zero. That is, if both nodes lie along a wall, the tilde velocity between these two nodes would also be set to zero. The reformulated continuity equation does not compute the change in tilde velocities in the same way. It is found that if the change in tilde velocities along the walls is not set equal to zero, the code becomes unstable. Therefore, to keep the reformulation stable, the code was modified so that the change in tilde velocities along walls during the inner iteration of the momentum equation is explicitly set to zero. This is accomplished as follows. A list of wall connections is computed upon initialization of the code, where a wall connection is determined when the `CONTAB` values of both wall nodes is zero. [See Appendix C for an explanation of the use of `CONTAB` in CHAD.] If both ends of a connection are wall nodes, then the connection is flagged as a connection between two wall nodes. Then, after the computation of the change in tilde velocities during the inner iteration of the momentum equation, a call is made to a new subroutine, `zero_wall_tildes`, that explicitly sets the tilde velocity components along connections flagged as connecting two wall nodes to zero.

3.4 Different Versions of the Continuity Equation

The continuity equation used in CHAD 5.0 does not match the continuity equation described in the physics manual for CHAD (O'Rourke, undated [a]). As

CHAD's physics manual was considered to be a rough draft (Sahota, 2001), this difference highlights the lack of adequate documentation available for the CHAD code that was described in Chapter 1.

The solution of the continuity equation occurs within the scalar GCR solver subroutine `solve_gcr_s`. In that subroutine, changes in pressure are computed. Those changes in pressure result in changes in density, nodal velocity and tilde velocities, which are all computed via the subroutine `res_lin_cont` and are required for the explicit recomputation of the linearized residual which is then used to determine convergence or to begin the next iteration for pressure. The documented form of the equation for the change in the cell-face tilde velocities is not the same as the coded form of the change in cell-face tilde velocities.

According to the physics manual, the change in tilde velocity is to be calculated using equation (3.8)

$$\delta \tilde{\mathbf{u}}_{\alpha} = \frac{\delta \mathbf{u}_v + \delta \mathbf{u}_{v\alpha}}{2} - \frac{\tilde{\Delta t}}{a^2 \rho_{\alpha}^n} \left\{ (\nabla \delta p)_{\alpha} - \frac{1}{2} [(\nabla \delta p)_v + (\nabla \delta p)_{v\alpha}] \right\} \quad (3.8)$$

which is computed in subroutine `cellface_tilde_change3` within CHAD. However, in the code, it is not computed in this fashion. Instead, in the CHAD code it is calculated as

$$\delta \tilde{\mathbf{u}}_{\alpha} = \frac{2\Delta t (\delta \nabla p)_{\alpha}}{a^2} \left(\frac{1}{\rho_v} + \frac{1}{\rho_{v\alpha}} \right) \quad (3.24)$$

which is computed in the subroutine (`cellface_tilde_change4`).

In the reformulated approach (Tzanos, 1998 [b]), the quantity $\delta\tilde{\mathbf{u}}_\alpha$ is to be evaluated using

$$\delta\tilde{\mathbf{u}}_\alpha = \left[\frac{\sum_\alpha C_{nb} \delta\mathbf{u}_{nb} + \delta Q_{u,v}}{C_v} \right] - \frac{\bar{V}_v}{C_v} \bar{i}_i (\delta\nabla p)_\alpha. \quad (3.25)$$

However, in CHAD_ANL code, the change in tilde velocity is actually computed using

$$\delta\tilde{\mathbf{u}}_\alpha = \frac{2\Delta t (\delta\nabla p)_\alpha}{a^2} \left(\frac{1}{\rho_v} + \frac{1}{\rho_{v\alpha}} \right) \quad (3.26)$$

where the PGS factor (a^2) has been added in by this author for clarity, as it had been deliberately removed from CHAD_ANL's source code at ANL. Note that equations (3.24) and (3.26) are identical.

Various tests conducted by the author indicate that CHAD converges with equation (3.26) but not with equation (3.25), and that is probably the reason why the use of equation (3.25), though proposed in reports, was abandoned in the coding stage.

In yet another approach to calculate $\delta\tilde{\mathbf{u}}_\alpha$, the change in tilde velocities can also be computed using

$$\delta\tilde{\mathbf{u}}_\alpha = -\frac{\bar{V}_v}{C_v} \bar{i}_i (\delta\nabla p)_\alpha. \quad (3.27)$$

This approach was tested in CHAD_ANL within the framework of the matrix formulation. Using equation (3.27) in place of equation (3.26) yields another formulation tested in the ANL version that is less stable than the formulation using equation (3.25),

converging for time steps of approximately $1/10^{\text{th}}$ the size used with equation (3.26) for the lid-driven cavity problem that is described thoroughly in Chapter 6.

Even though equations (3.24), (3.26) and (3.27) do not require the use of nodal velocities that equation (3.8) requires, it should be noted that the change in nodal velocity is still required in the computation of the linear residual. A more explicit representation of equation (3.6) explicitly shows the use of nodal velocities in computing flow leaving the open boundaries in the domain

$$\begin{aligned} (R_p)_{v,lin} = & (R_p)_v + \delta\rho_v + \\ & \frac{\Delta t}{V_v^{n+1}} \left[\sum_{\alpha} \rho_{\alpha} \delta \tilde{\mathbf{u}}_{\alpha} \cdot \mathbf{A}_{\alpha}^{n+1} + \sum_{\alpha} \delta \rho_{\alpha} \left(\tilde{\mathbf{u}}_{\alpha} \cdot \mathbf{A}_{\alpha}^{n+1} - \frac{\phi_{\alpha}}{\Delta t} \right) - \sum_v \rho_{v,out} \delta \mathbf{u} \cdot \mathbf{A}_{v,open} - \sum_v \delta \rho_v \cdot \mathbf{A}_{v,open} \right] \end{aligned} \quad (3.28)$$

For problems that are nearly incompressible, equation (3.28) simplifies to

$$(R_p)_{v,lin} = (R_p)_v + \frac{\Delta t}{V_v^{n+1}} \left[\sum_{\alpha} \rho_{\alpha} \delta \tilde{\mathbf{u}}_{\alpha} \cdot \mathbf{A}_{\alpha}^{n+1} - \sum_v \rho_{v,out} \delta \mathbf{u} \cdot \mathbf{A}_{v,open} \right] \quad (3.29)$$

while in the case of closed-domain problems such as the driven cavity flow that we explore later, it simplifies further to

$$(R_p)_{v,lin} = (R_p)_v + \frac{\Delta t}{V_v^{n+1}} \left[\sum_{\alpha} \rho_{\alpha} \delta \tilde{\mathbf{u}}_{\alpha} \cdot \mathbf{A}_{\alpha}^{n+1} \right]. \quad (3.30)$$

In summary, four methods exist in CHAD for computing changes in tilde velocities, and they depend on the variables supplied. These are summarized in Table 3.1.

In CHAD 5.0, the change in temperature due to changes in pressure does not follow equation (3.13) as stated in the physics manual. Instead, it is evaluated as

$$\delta T_v = \left(\min \left(\frac{1 - \rho_v \frac{\partial h}{\partial p} \Big|_{T,v}^n}{\rho_v \frac{\partial h}{\partial p} \Big|_{T,v}^n}, 0.75 \frac{\frac{\partial \rho}{\partial p} \Big|_{T,v}^n}{\frac{\partial \rho}{\partial T} \Big|_{p,v}^n} \right) \right) \delta p_v. \quad (3.31)$$

The second term in the **min** bracket is required because density changes during the solution of the energy equation. However, for predominantly incompressible flows—which are the main focus of this dissertation—changes in density are small. This change in δT_v from the documented form to the coded form is significant because it no longer depends on nodal velocities. Therefore, we can insert equation (3.31) into equation (3.7) to get

$$\delta \rho_v = \frac{\partial \rho}{\partial p} \Big|_{T,v}^n \delta p_v + \frac{\partial \rho}{\partial T} \Big|_{p,v}^n \delta T_v = \left(\frac{1}{R_{gas} T} + \frac{\partial \rho}{\partial T} \Big|_{p,v}^n \frac{1 - \rho_v \frac{\partial h}{\partial p} \Big|_{T,v}^n}{\rho_v \frac{\partial h}{\partial p} \Big|_{T,v}^n} \right) \delta p_v. \quad (3.32)$$

Equation (3.32) can now be included in a matrix formulation since it directly depends on a change in nodal pressure (pressure we are already calculating). By including (3.32) into the matrix we have a system of equations that, to some extent, predicts the changes due to compressibility effects (however small we may expect them to be). Therefore, we now can predict the direct change in density, the direct change in mass flow due to changes in density, and the direct change in mass due to changes in cell-face tilde velocities.

In the reformulated scheme, the change in density at the median mesh was evaluated using

$$\delta\rho_\alpha = \frac{1}{2}(\delta\rho_v(1+VFX) + \delta\rho_{v\alpha}(1-VFX)). \quad (3.33)$$

Here the change in density depends upon upwind direction indicators (VFX). At this time the change in median mesh density due to changes in pressure has not been computed and is not included in the PETSc formulation based upon the reformulated continuity equation.

3.5 Solving the Continuity Equation Using GCR

The continuity equation is solved using a matrix-free formulation of the GCR method that is implemented in the `solve_gcr_s` subroutine. `solve_gcr_s` is a generic scalar GCR solver routine based on an explicit computation of the scalar equation linear residuals. Convergence within `solve_gcr_s` indicates convergence of the scalar equation. Convergence is checked before the first, and during every iteration of the GCR algorithm. The initial convergence check occurs when the nonlinear residual is first copied to the linear residual. This “linear” residual is checked for convergence.

There are three test cases to which the “linear” residual is applied; any one of which if satisfied would flag convergence. Using the continuity equation as an example, the first condition occurs when

Table 3.1. Change in tilde velocity computations available within CHAD 5.0

Subroutine	Input Variables compute tilde velocities given	Formulation
cellface _tilde _change1 called by cellface _tilde1	<i>given changes in pressure and velocity</i> <ul style="list-style-type: none"> • deltap, at nodes • deltap, at termini • delta-velocity, at termini • old-density, at nodes • old-density, at termini • old-speed of sound, at termini 	$-\Delta t \left(\frac{2}{a^2} \frac{(\nabla p)_\alpha}{(\rho_v + \rho_{v\alpha})} - \frac{1}{2} \left[\left(\frac{(\nabla p)_v}{\rho_v} \right)_v + \left(\frac{(\nabla p)_v}{\rho_v} \right)_{v\alpha} \right] \right)$
cellface_tilde _change2 called by (no calling routine)	<i>given changes in pressure, nodal pressure gradient</i> <ul style="list-style-type: none"> • deltap, at termini • change in gradient of pressure, at nodes • old-density, at nodes • old-density, at termini 	$-\Delta t \left(\frac{2}{a^2} \frac{(\nabla p)_\alpha}{(\rho_v + \rho_{v\alpha})} - \frac{1}{2} \left[\left(\frac{(\nabla p)_v}{\rho_v} \right)_v + \left(\frac{(\nabla p)_v}{\rho_v} \right)_{v\alpha} \right] \right)$
cellface _tilde _change3 called by (no calling routine)	<i>given changes in pressure, velocity, nodal pressure gradient</i> <ul style="list-style-type: none"> • deltap, at termini • delta-velocity, at termini • change in gradient of pressure, at nodes • old-density, at nodes • old-density, at termini 	$\frac{\Delta u_v + \Delta u_{v\alpha}}{2} - \Delta t \left(\frac{2}{a^2} \frac{(\nabla p)_\alpha}{(\rho_v + \rho_{v\alpha})} - \frac{(\nabla p)_v}{\rho_v} \right)$
cellface_tilde _change4 called by res_lin_cont	<i>given changes in pressure</i> <ul style="list-style-type: none"> • deltap, at termini • old-density, at termini 	$\delta \tilde{u}_\alpha = \frac{2\Delta t (\delta \nabla p)_\alpha}{a^2} \left(\frac{1}{\rho_v} + \frac{1}{\rho_{v\alpha}} \right)$

$$\left| \left(R_p \right)_{v,lin} \right| \leq 0.1 \left| \left(R_p \right)_v \right| \text{ for all } v \quad (3.34)$$

where the sidebars indicate absolute values and not vector norms. The second condition that indicates inner iteration convergence is when

$$\left| \left(R_p \right)_{v,lin} \right| \leq 10^{-8} \rho_v \text{ for all } v \quad (3.35)$$

Convergence is also deemed achieved (third condition) when the change between two successive inner iterations is

$$\left| \delta(\delta p_v) \right| \leq 0.1 \left(10^{-3} * \max \left(\max(p) - \min(p), \max \left(\frac{1}{2} \rho u^2 \right) \right) \right) \text{ for all } v. \quad (3.36)$$

Equation (3.35) is the equation listed in the physics manual (O'Rourke, undated [a]), though in the actual code, it is found to be

$$\left| \left(R_p \right)_{v,lin} \right| \leq 10^{-10} * (\max(p) - \min(p)) \text{ for all } v. \quad (3.37)$$

If any of these three conditions are satisfied in the outer iteration loop, the code exits the subroutine without performing any iterations. If outer iteration is not converged, the system is solved using a matrix free implementation of the GCR algorithm given by

$$\begin{aligned} \alpha_k &= \frac{(r^k)^T (Ap_k)}{(Ap_k)^T (Ap_k)} \\ x^{k+1} &= x^k + \alpha_k p_k \\ r^{k+1} &= r^k - \alpha_k Ap_k \\ p_{k+1} &= r^{k+1} - \sum_{j=0}^k \frac{(Ar^{k+1})^T (Ap_j)}{(Ap_j)^T (Ap_j)} p_j \end{aligned} \quad (3.38)$$

Here, α is a multiplier for a search direction vector, p_k . Since only matrix-vector products exist, this algorithm is written in matrix-free form, thereby saving memory when programmed into CHAD.

In CHAD, the linear residual is computed directly using equation (3.6). To compute the residual, changes in tilde velocities, temperature, nodal and cell-face density are all required (equations 3.7, 3.8, 3.9, and 3.12). The new linearized residual is computed, and checked for convergence. The recomputed tilde velocities, temperatures, and density terms are *not* used to modify the system of equations. They simply produce the new linear residual required by GCR. The new residual is used within the GCR iterative cycle, recomputed during the iterations, until the computed linear residual is low enough to satisfy convergence requirements or until the maximum number of iterations allowed is reached.

After one of the above mentioned stopping criteria is met, limitations that arise due to the linearized nature of the equation of state must be applied to density, temperature and pressure. After computing the final changes in pressure and applying limits on the variables that depend on pressure, GCR exits and returns to the `solve_or_res_nl_cont` routine. It is in this routine that the changes in velocities, temperatures, density and pressure are used to update velocity, temperature, density, and pressure.

3.5.1 Preconditioning GCR

In the solution of the scalar equations solved using the GCR algorithm, the reciprocal of the derivative of the residual of the continuity equation with respect to pressure `RDRESCDP` is used to precondition the system of equations. It is given by

$$d_v = \left[\frac{\partial (R_p)_{v,lin}}{\partial p_v} \right]^{-1} \quad (3.39)$$

and it serves as a simple, diagonal preconditioner. Within CHAD 5.0, this is given by

$$d_v = \left[\left. \frac{\partial \rho}{\partial p} \right|_{T,v}^n + \left. \frac{\partial \rho}{\partial T} \right|_{p,v}^n \delta T_v \right]^{-1}. \quad (3.40)$$

It should be apparent that for incompressible flows, equation (3.40) is extremely vulnerable to division by zero which will cause numerical difficulties in the solver. In reality, division by zero does not exactly occur as a trap is set which prevents division by zero and takes the form

$$\max \left(\left| \left[\left. \frac{\partial \rho}{\partial p} \right|_{T,v}^n + \left. \frac{\partial \rho}{\partial T} \right|_{p,v}^n \delta T_v \right]^{-1} \right|, \text{tinynum} \right) \quad (3.41)$$

where `tinynum` is a very small number close to the size of the smallest exponent of machine precision and is the dominant term in purely incompressible flows. On the *Reserv* computing cluster at Argonne National Laboratory d_v is approximately 10^{+207} . Even though the use of `tinynum` will not result in a FORTRAN exception error due to division by zero, the sheer magnitude of the preconditioner will result in numerical difficulties for many problems. In chapter 7, we modify equation (3.40) to handle an incompressible equation of state.

In this chapter, we reviewed the reformulation of the continuity equation that was implemented within CHAD_ANL. This reformulated continuity equation and associated code was transferred into structurally different CHAD 5.0 to form a new version of CHAD we refer to as CHAD_UI. In doing so, a number of differences between CHAD_ANL, CHAD 5.0 and the documented forms of these equations were discovered. Significant differences were discussed here. The solution to the continuity equations using the GCR solver was then described, including CHAD's simple preconditioning and convergence criteria. In the next chapter, we describe the construction of a matrix system to solve the continuity equation suitable for use in PETSc.

Chapter 4: Implementing PETSc into CHAD

4.1 An Overview of PETSc

PETSc is a software library for efficient parallel solution of sets of algebraic equations that result, for example, from the discretization of partial differential equations. In particular, PETSc provides efficient parallel vector and matrix manipulation as well as parallel linear and nonlinear equation solvers (Balay, 2000). PETSc provides a set of functions and subroutines that can be *called* from a software package. Users programmatically supply data from their application to specific PETSc subroutines, which then operate on the data and return solutions to the calling program. These PETSc subroutines are specifically designed to be portable and highly efficient for parallel computers, while simultaneously removing the programmer from the complexities of implementing new solvers as well as the details of parallel algorithms and computing.

PETSc requires three other software packages; BLAS (Basic Linear Algebra Set), LAPACK (Linear Algebra Package), and an implementation of MPI (Message Passing Interface). A serial version of MPI is included with PETSc should a user wish to run PETSc using a single processor.

Implementing PETSc in a CFD code such as CHAD is a straightforward, but non-trivial process. First, PETSc must be initialized prior to use, at which time it configures itself at runtime with MPI. After initialization, for example, if the solution of a set of equations of the form $\underline{\underline{A}}\underline{x} = \underline{b}$ is desired, specific PETSc subroutines are called to first

create an *instance* of the matrix $\underline{\underline{A}}$, and two vectors \underline{x} and \underline{b} . Next, these instances are filled with actual data provided by the user as subroutine arguments. Then, a solver *instance* is created. This solver *instance* is filled with the matrix $\underline{\underline{A}}$, a preconditioner, and a right-hand-side vector \underline{b} , as well as an empty solution vector \underline{x} . PETSc solves this system via a method specified by the user (or by a default method when no method is specified) and the results are copied into the empty solution vector which is returned to the calling program, in this case, CHAD. The calling program (CHAD) then uses that data just as if it had been obtained via its own solvers. When PETSc is no longer needed, it is terminated gracefully.

PETSc offers two major advantages to its users. First, a large number of solvers no longer need to be programmed into every new application. PETSc, for example, currently offers fifteen Kyrlov methods including GMRES (Saad and Schultz, 1986), Stabilized Bi-Conjugate Gradient and conjugate residual solvers. Additional solvers are constantly being added and improvements are made to existing solvers, which are made available in newer versions of PETSc. A second and probably more significant advantage of PETSc, is that its solvers are designed for optimal solutions on parallel computers. Therefore, a code developer can focus on numerical schemes and other aspects of their code without worrying about continuously updating their solver or chasing the latest paradigm in parallel computing.

PETSc has previously been used with success in CFD. In one application, PETSc was used to increase the performance of a NASA CFD code called Fun-3D (Gropp,

2001), while in another it is used to improve the performance of an implicit, unstructured, time-dependent CFD code (Karimian, 2005.) It has been used successfully in automotive applications (Franck, 2004) as well as naval hydrodynamics (Paterson, 2004.) PETSc is often used in massively parallel simulations and computations (Keyes, 2001, Knepley, 1999)

Within the scope of this dissertation, PETSc's primary use will be to supplement the current continuity equation solver in CHAD with a PETSc matrix solver of the form $\underline{\underline{A}}\underline{x} = \underline{b}$. Currently, a typical thermal-hydraulic problem solved using CHAD spends a large fraction of its time on the solution of the continuity equation—approximately 35%. It is believed that with the special attention PETSc pays to the parallel construction and solution of systems of equations, the performance of the continuity equation in CHAD will improve.

Some work has previously been done to allow the use of PETSc routines in CHAD. In 1998 a Newton-Krylov scheme was implemented into CHAD for the solution of both the continuity and momentum equations (Canfield-c, 2003). Results failed to show significant improvement in the solution times of tested problems. Nevertheless, it was believed that more significant performance gains could be achieved through proper preconditioning of the solver (Canfield-c).

4.2 The Matrix Equations

At the heart of the PETSc package is the linear solver package which uses Krylov subspace methods to obtain solutions to equations of the form

$$\underline{\underline{A}}\underline{x} = \underline{b}. \quad (4.1)$$

Krylov subspace methods are a type of iterative solution algorithm in which optimal approximations to equation (4.1) are obtained with only a modest amount of work. Starting with an initial guess for the unknown vector \underline{x} , successive guesses for the unknown vector are obtained from the space spanned by \underline{b} and $\underline{\underline{A}}\underline{b}$ which is given by (Greenbaum, 1997)

$$\underline{x} \in \text{span}\{\underline{b}, \underline{\underline{A}}\underline{b}\}. \quad (4.2)$$

Each iteration of the Krylov space proceeds so that the approximation at the k^{th} iteration is given by

$$\underline{x}_k \in \text{span}\{\underline{b}, \underline{\underline{A}}\underline{b}, \dots, \underline{\underline{A}}^{k-1}\underline{b}\} \quad k = 1, 2, \dots \quad (4.3)$$

where the space spanned in equation (4.3) is the *Krylov subspace*.

To solve CHAD's continuity equation for pressure using the matrix formulation given by equation (4.1), the system

$$\left(R_p\right)_{v,lin} = R_{p,v} + \frac{\partial \left(R_p\right)_{v,lin}}{\partial p} \delta p \equiv 0 \quad (4.4)$$

is constructed and solved as

$$\underbrace{\frac{\partial \left(R_p\right)_{v,lin}}{\partial p}}_{\underline{\underline{A}}} \underbrace{\delta p_v}_{\underline{x}} = \underbrace{-R_{p,v}}_{\underline{b}}. \quad (4.5)$$

where the linearized terms on the left hand side have been separated from the constants on the right hand side. Inserting the derivative of equation (3.6) into equation (4.5) gives

$$\begin{aligned} \delta\rho_v + \frac{\Delta t}{V_v^{n+1}} \sum_{\alpha} \left[\delta\rho_{\alpha} \left(\tilde{\mathbf{u}}_{\alpha} \cdot \mathbf{A}_{\alpha}^{n+1} - \frac{\phi_{\alpha}}{\Delta t} \right) + \rho_{\alpha} \delta\tilde{\mathbf{u}}_{\alpha} \cdot \mathbf{A}_{\alpha}^{n+1} \right] \\ - \frac{\Delta t}{V_v^{n+1}} \left(\rho_{v,out} \delta\mathbf{u} \cdot \mathbf{A}_{v,open} + \delta\rho_v \mathbf{u} \cdot \mathbf{A}_{v,open} \right) = - \left(R_p \right)_v \end{aligned} \quad (4.6)$$

The first term in equation (4.6) represents the change in nodal mass that results from changes in pressure due to the equation of state. The second term represents changes in mass entering or leaving a cell due to changes in cell-face density. The third term accounts for changes in mass due to a moving mesh sweeping out a volume. The fourth term signifies changes in cell mass due to changes in the cell-face velocities that arise due to changes in pressure. In incompressible flow, this is the dominant term. The fifth and sixth terms account for changes in mass at outflow nodes due to changes in nodal velocity and cell density, respectively.

The terms in equation (4.6) depend on changes in pressure at nodes and cell-faces. However, the cell-face pressures are obtained from nodal pressures using communication across connections. We leave cell-face terms in connection notation for the sake of clarity.

The diagonal matrix element for each node, A_v , will contain the coefficient for changes in cell mass due to the equation of state plus the coefficients for each of the cell-face for that node.

$$\begin{aligned}
A_v &= \delta\rho_v + \frac{\Delta t}{V_v^{n+1}} \sum_{\alpha} \left[\delta\rho_{\alpha} \left(\tilde{\mathbf{u}}_{\alpha} \cdot \mathbf{A}_{\alpha}^{n+1} - \frac{\phi_{\alpha}}{\Delta t} \right) + \rho_{\alpha} \delta\tilde{\mathbf{u}}_{\alpha} \cdot \mathbf{A}_{\alpha}^{n+1} \right] \\
&\quad - \frac{\Delta t}{V_v^{n+1}} \left(\rho_{v,out} \delta\mathbf{u}_v \cdot \mathbf{A}_{v,open} + \delta\rho_v \mathbf{u}_v \cdot \mathbf{A}_{v,open} \right) \\
&\quad \text{where} \\
&\quad \frac{\delta\mathbf{u}}{\partial p_v} = -\frac{\Delta t}{a^2 \rho_v V_v^{n+1}} \sum_{\alpha} \mathbf{A}_{p,\alpha}^{n+1} \\
&\quad \frac{\delta\tilde{\mathbf{u}}_{\alpha}}{\partial p_v} = \frac{2\Delta t}{a^2} \left(\frac{1}{\rho_v} + \frac{1}{\rho_{v\alpha}} \right) \frac{d_{a,\alpha}}{2|da|^2} \\
&\quad \frac{\delta\rho_v}{\partial p_v} = \left(\frac{1}{R_{gas}T} + \frac{\partial\rho}{\partial T} \Big|_{p,v} \left(\frac{1 - \rho_v \frac{\partial h}{\partial p} \Big|_{T,v}^n}{\rho_v \frac{\partial h}{\partial p} \Big|_{T,v}^n} \right) \right) \\
&\quad \frac{\delta\rho_{\alpha}}{\partial p_v} = \begin{cases} \left(1 - \frac{1}{C_{\alpha}} \right) \frac{\delta\rho_v}{\partial p_v} & \text{if } C_{\alpha} > 1 \text{ and } \mathbf{u}_{rel} \cdot \mathbf{A}_{\alpha}^{n+1} > 0 \\ \left(1 - \frac{1}{C_{\alpha}} \right) \frac{\delta\rho_{v\alpha}}{\partial p_v} & \text{if } C_{\alpha} > 1 \text{ and } \mathbf{u}_{rel} \cdot \mathbf{A}_{\alpha}^{n+1} < 0 \\ 0 & \text{otherwise} \end{cases}
\end{aligned} \tag{4.7}$$

The off-diagonal matrix elements for each node, $A_{v\alpha}$, will contain the coefficient for changes in cell mass due to changes in the mass flux through each face. The number of off-diagonal terms in the matrix is determined by the cell type, which in the case of a hexahedral cell is six, five for a wedge, and four in the case of a tetrahedral cell. For essentially all cases, the resulting matrix will be sparse. Additionally, when the grid is structured, the matrix will be banded.

$$\begin{aligned}
A_{v\alpha} &= -\frac{\Delta t}{V_v^{n+1}} \sum_{\alpha} \left[\delta \rho_{\alpha} \left(\tilde{\mathbf{u}}_{\alpha} \cdot \mathbf{A}_{\alpha}^{n+1} - \frac{\phi_{\alpha}}{\Delta t} \right) + \rho_{\alpha} \delta \tilde{\mathbf{u}}_{\alpha} \cdot \mathbf{A}_{\alpha}^{n+1} \right] \\
&\quad \text{where} \\
\frac{\delta \tilde{\mathbf{u}}_{\alpha}}{\partial p_{\alpha}} &= \frac{2\Delta t}{a^2} \left(\frac{1}{\rho_v} + \frac{1}{\rho_{v\alpha}} \right) \frac{d_{a,\alpha}}{2|da|^2} \\
\frac{\delta \rho_v}{\partial p_v} &= \left(\frac{1}{R_{gas}T} + \frac{\partial \rho}{\partial T} \Big|_{p,v} \left(\frac{1 - \rho_v \frac{\partial h}{\partial p} \Big|_{T,v}^n}{\rho_v \frac{\partial h}{\partial p} \Big|_{T,v}^n} \right) \right) \\
\frac{\delta \rho_{\alpha}}{\partial p_v} &= \begin{cases} \left(1 - \frac{1}{C_{\alpha}} \right) \frac{\delta \rho_v}{\partial p_v} & \text{if } C_{\alpha} > 1 \text{ and } \mathbf{u}_{rel} \cdot \mathbf{A}_{\alpha}^{n+1} > 0 \\ \left(1 - \frac{1}{C_{\alpha}} \right) \frac{\delta \rho_{v\alpha}}{\partial p_v} & \text{if } C_{\alpha} > 1 \text{ and } \mathbf{u}_{rel} \cdot \mathbf{A}_{\alpha}^{n+1} < 0 \\ 0 & \text{otherwise} \end{cases} \quad (4.8)
\end{aligned}$$

The computed changes in nodal pressure, δp_v are stored in the unknown vector,

\underline{x}

$$\underline{x} = \delta p_v \quad (4.9)$$

The residual vector, \underline{b} , is given by

$$b_v = \rho_v - \rho_v^{old} - \frac{\Delta t}{V_v^{n+1}} \left[\sum_{\alpha} \rho_{\alpha} \left(\tilde{\mathbf{u}}_{\alpha} \cdot \mathbf{A}_{\alpha}^{n+1} \right) \right]. \quad (4.10)$$

When the reformulated continuity equation is used, the unknown and residual vectors remain the same, but the coefficient matrix is different. In this case, the diagonal terms in the coefficient matrix are given by

$$\begin{aligned}
A_v &= \delta\rho_v + \frac{\Delta t}{V_v^{n+1}} \sum_{\alpha} \left[\delta\rho_{\alpha} \left(\tilde{\mathbf{u}}_{\alpha} \cdot \mathbf{A}_{\alpha}^{n+1} - \frac{\phi_{\alpha}}{\Delta t} \right) + \rho_{\alpha} \delta\tilde{\mathbf{u}}_{\alpha} \cdot \mathbf{A}_{\alpha}^{n+1} \right] \\
&\quad - \frac{\Delta t}{V_v^{n+1}} \left(\rho_{v,out} \delta\mathbf{u}_v \cdot \mathbf{A}_{v,open} + \delta\rho_v \mathbf{u}_v \cdot \mathbf{A}_{v,open} \right) \\
&\quad \text{where} \\
&\quad \frac{\delta\mathbf{u}}{\partial p_v} = -\frac{\Delta t}{a^2 \rho_v V_v^{n+1}} \sum_{\alpha} \mathbf{A}_{p,\alpha}^{n+1} \\
&\quad \frac{\delta\tilde{\mathbf{u}}_{\alpha}}{\partial p_{\alpha}} \approx \frac{\bar{V}_v}{C_v} \frac{d_{a,\alpha}}{2|da|^2} \\
&\quad \frac{\delta\rho_v}{\partial p_v} = \left(\frac{1}{R_{gas}T} + \frac{\partial\rho}{\partial T} \Big|_{p,v} \left(\frac{1 - \rho_v \frac{\partial h}{\partial p} \Big|_{T,v}^n}{\rho_v \frac{\partial h}{\partial p} \Big|_{T,v}^n} \right) \right) \\
&\quad \frac{\delta\rho_{\alpha}}{\partial p_v} = \frac{1}{2} \left[\delta\rho_v (1 + VFX) + \delta\rho_{v\alpha} (1 - VFX) \right]
\end{aligned} \tag{4.11}$$

The term $\frac{d_{a,\alpha}}{2|da|^2}$ is an approximation to the average cell length used in the computation

of the cell-face pressure gradient. VFX is the first-order upwind direction indicator, and

$$\begin{aligned}
A_{v\alpha} &= -\frac{\Delta t}{V_v^{n+1}} \sum_{\alpha} \left[\delta\rho_{\alpha} \left(\tilde{\mathbf{u}}_{\alpha} \cdot \mathbf{A}_{\alpha}^{n+1} - \frac{\phi_{\alpha}}{\Delta t} \right) + \rho_{\alpha} \delta\tilde{\mathbf{u}}_{\alpha} \cdot \mathbf{A}_{\alpha}^{n+1} \right] \\
&\quad \text{where} \\
&\quad \frac{\delta\tilde{\mathbf{u}}_{\alpha}}{\partial p_{\alpha}} = \frac{\bar{V}_v}{C_v} \frac{d_{a,\alpha}}{2|da|^2} \\
&\quad \frac{\delta\rho_v}{\partial p_v} = \left(\frac{1}{R_{gas}T} + \frac{\partial\rho}{\partial T} \Big|_{p,v} \left(\frac{1 - \rho_v \frac{\partial h}{\partial p} \Big|_{T,v}^n}{\rho_v \frac{\partial h}{\partial p} \Big|_{T,v}^n} \right) \right) \\
&\quad \frac{\delta\rho_{\alpha}}{\partial p_v} = \frac{1}{2} \left[\delta\rho_v (1 + VFX) + \delta\rho_{v\alpha} (1 - VFX) \right]
\end{aligned} \tag{4.12}$$

4.3 Final Convergence and Variable Updates

In the GCR algorithm, when any of the three convergence criteria given by equations (3.34 – 3.36) is satisfied, the changes in the variables given by equations (3.7 – 3.12) are evaluated one last time. This computation differs slightly from the computation that was used to deduce convergence in that limits for this computation are placed on the changes in pressure. These limits are computed prior to the call to the `solve_gcr_s` routine, and are stored in the `DELTAPMIN` and `DELTAPMAX` arrays. Additionally, limits are placed upon the maximum allowable changes in velocity, temperature, pressure and density.

In solving the continuity equation using the PETSc matrix solver, limits are placed upon pressure in exactly the same way as they are done with the GCR solver. In fact, the changes in velocity, temperature, and density are computed using exactly the same function call to `res_lin_cont` as was done with the GCR solver.

Equation (4.13) shows the limit placed on the u component of velocity, in vector form. Similar equations exist for the v and w components of velocity as well as the change in temperature

$$\delta u_v = \text{sign}(\min(|\delta u_v|, 0.25 \cdot |u_v|), \delta u_v). \quad (4.13)$$

Changes in nodal density are governed by

$$\delta \rho_n = \min(\max(\delta \rho_n, -0.9 \cdot \rho_n), 10 \cdot \rho_n) \quad (4.14)$$

and changes in cell-face density are given by

$$\delta\rho_\alpha = \mathbf{max}(\delta\rho_\alpha, -0.5 \cdot \rho_\alpha). \quad (4.15)$$

The limits on the change in pressure are quite complicated and are not shown here. However, they are set so that they allow for a maximum of ~25% change in temperature and density so as to ensure a positive pressure for ideal gases. Changes larger than 25% cannot be ensured to guarantee positive pressure and would violate the linearized equation of state anyway. No limits are applied to the cell-face velocities.

Though limits are set on changes in variables, the reader is cautioned against thinking of these limits as under-relaxation factors. It is possible for a solution to remain within the imposed limits and therefore not require any changes during the solution to the continuity equation, whereas in under-relaxation these limits are *always* applied. That said, under relaxation capability is not built in the numerical algorithm in CHAD.

In the next chapter, the actual PETSc implementation is described in detail.

Chapter 5: Programming PETSc into CHAD

5.1 Preparing CHAD for PETSc

Several modifications are required in CHAD to seamlessly integrate it with PETSc. In particular, memory is required to store the variables for the coefficient matrix $\underline{\underline{A}}$ and residual vector \underline{b} , which requires declaration, allocation, and in some cases, computation. These variables were originally developed as part of an incomplete implementation of a different matrix package developed by Tzanos within CHAD_ANL (Tzanos-b). As CHAD 3.0 evolved to later versions, migrating the matrix formulation features of CHAD_ANL to the most recent versions of CHAD required significant alterations in the declarations and scopes of the variables. The results of those changes are briefly described below.

The coefficient matrix, unknown vector, and the residual vector are required components of the PETSc solver. The coefficient matrix construction is split into two parts for reasons which will be described shortly. The first part contains the diagonal terms, designated as $A0$. The second part contains the off-diagonal terms designated as $A0T$. Two arrays are used instead of one, primarily due to the particular data structures being defined. However, an incidental benefit to using a separate array to store the diagonal terms results when an accurate representation of the diagonal term is desired to compute the preconditioner for the GCR solver. This is discussed more thoroughly in Chapter 7.

The residual vector is declared as `BB`, while the unknown vector is already provided by CHAD for the default GCR solver and is declared as `DELTAP`. All other variables are already declared in CHAD with the exception of temporary work variables and the matrix coefficients. Those coefficients, `CNUM`, `CNVM`, and `CNWM` are required for the implementation of the reformulated continuity equation and are not declared for the exclusive use of the PETSc solver. They are described by (Tzanos-b).

The terms used to store the matrix and the residual vector are declared in `solve_cont_module_c`, with diagonal terms of a size equal to the number of nodes resident on a processor, and off diagonal terms equal to the number of connections resident on that processor, with two values per connection. The terms used to compute the pressure coefficient (`CNUM`, `CNVM`, `CNWM`) in the reformulated continuity equation are defined in `hydro_module_c` instead of in `solve_cont_module_c` because they must be accessed by the cell-face routines which reside outside of the solver portions of the code, while `solve_cont_module_c` is only intended to be accessed by the continuity solver portion of the code. Table 5.1 summarizes these variables and their locations.

Table 5.1: New CHAD Variables Required for the PETSc Implementation

Variable	Purpose	Location	Scope	Size
CNUM	pressure coefficient	hydro_module_c	all hydro routines	# connections
A0	diagonal matrix entry	solve_cont_module_c	continuity equation	# nodes
A0T	off-diagonal matrix entry	solve_cont_module_c	continuity equation	2, # connections
BB	residual vector	solve_cont_module_c	continuity equation	# nodes

5.2 Initializing PETSc

PETSc must be *initialized* before it can be used. It is initialized by calling a PETSc initialization routine. In CHAD, this initialization has been accomplished through a call to a new subroutine (called `petsc_chad_init`) from CHAD's main driving routine (`chad_main`). The call occurs right after the input file is read via a call to the CHAD subroutine that reads the input file (`read_input`). It is called only if the input variable that specifies the use of the PETSc matrix solver (`matrix_solve`) is found to be logically true (`.true.`) in the input file. Use of the matrix solver with the reformulated continuity equation requires that the flags controlling the options for the first order upwinding of density, momentum (`iup_rho` and `iup_uvw`), as well as the flag specifying the usage of the reformulated continuity equation (`ictz`), all be set to one (1) in the input file. To enable the PETSc solver with the original continuity equation, the user must set a new variable (`matrix_osolve`) to logically true (`.true.`) in the input file.

Within the new subroutine which is used to initialize PETSc (`petsc_chad_init`), PETSc subroutine `petscinitialize` is called

```
petscinitialize(PETSC_NULL_CHARACTER, IERR)
```

where the final argument of the call (`IERR`) returns any error code. `IERR` is the trailing argument in all PETSc calls made using FORTRAN. The other argument of the call, (`PETSC_NULL_CHARACTER`) is a PETSc implemented FORTRAN *null* and is included within the routine via `include` files. Once PETSc has been initialized using the above

subroutine call, PETSc remains active within any portion of the code until it is explicitly shut down, a process described later in this chapter.

Implementation of the PETSc solver in CHAD is designed to complement the generalized conjugant gradient-scalar (gcr_s) solver within CHAD. As such, when the matrix based approach is desired (i.e. if either `matrix_solv` or `matrix_osolv` are `.true.`) the routine that drives the solution of the continuity equation (`solve_or_res_nl_cont`) bypasses the scalar solver (`solve_gcr_s`) and in its place calls a parallelized PETSc based solver (`solve_petsc_cont_par_2`). While a similar subroutine exists for serial simulations (`solve_petsc_cont`), access to this serial method has not been provided.

Variables required for CHAD's implementation of PETSc, most of which consist of data types specific to PETSc, are initialized upon entering the parallel PETSc solver routine added to CHAD (`solve_petsc_cont_par_2`). While the names of many of the variable types may be obvious to the reader, because some may not be, they are all described in Table 5.2.

Table 5.2: Additional Variables Required for the PETSc Implementation

Variable	Purpose	Type	Size
I_P	PETSc array storing unknown vector	offset	N/A
Mat_A	matrix	matrix	nnodemaxtot, nnodemaxtot
P_ARRAY	index of array for I_P	scalar	array(1)
PTC_B	residual vector	vector	nnodemaxtot
PTC_P	unknown vector	vector	nnodemaxtot
SLES	linear solver context	sles	N/A
TMP	temporary scalar	scalar	rank 0

After variable declarations, data structures are created that are filled later and used to solve the system of equations. A PETSc parallel vector can be constructed using the call

```
VecCreateMPI(PETSC_COMM_WORLD, PETSC_DECIDE, nnodemaxtot,
             PTC_P, IERR)
```

where the MPI communicator is given by (PETSC_COMM_WORLD) and is provided by PETSc when PETSc is initialized. The local vector size on each processor is computed by PETSc using the variable (PETSC_DECIDE) when the global size of the vector (nnodemaxtot) is provided—as in the case shown above. The fourth term in the variable list specifies the name of the vector being created (PTC_P), which in this case is used to store the unknowns vector \underline{x} and will be filled with the solution terms δp_v .

The total number of nodes in the problem (nnodemaxtot) is not a global variable, so it is computed before the PETSc solver routine is called from CHAD. As CHAD uses a distributed memory architecture, each processor operates on its own portion of the global data set. Thus, while each processor owns a number of nodes given by (nnodemax), the total number of nodes in the entire problem is the sum of all nodes over the domain (scalar_sum(nnodemax)). The computation of (nnodemaxtot) occurs before the call to (solve_petsc_cont_par_ori), and the value is passed as an argument to the routine. At a later time, this variable may be made global.

Specifying the global size of a vector instructs PETSc to determine the local size of the vectors residing on each processor. The opposite can also be true; specifying the

local size of the vectors instructs PETSc to determine the global size of the vector, as is shown below.

```
VecCreateMPI(PETSC_COMM_WORLD, nnodemax, PETSC_DETERMINE,
             PTC_P, IERR)
```

When local vector sizes are provided, the variable (PETSC_DETERMINE) instructs PETSc to calculate the global vector length. Both methods are provided in CHAD with the second method being the default, but only the second method will work when the grid partitioning is provided as part of the mesh file.

After creating the vector, it is then “set” to utilize default settings using the routine

```
VecSetFromOptions(PTC_P, IERR).
```

Next, a new vector \underline{b} is created to store the right hand side of the system (PTC_B).

When grid partitioning is not provided in the mesh file, it can be created using identical settings as the first vector using the vector duplication routine

```
VecDuplicate(PTC_P, PTC_B, IERR).
```

This process is faster than creating a new vector from scratch and is therefore used in place of creating a new vector and setting its options. When grid partitioning information is provided, the \underline{b} vector is created as before using the routine `VecCreateMpi`.

A matrix is constructed, in a way analogous to building vectors, using the routine

```
MatCreate(PETSC_COMM_WORLD, m, n, M, N, Mat_A, IERR)
```

where m and n are the local sizes of the rows and columns of the matrix, while M and N represent global sizes for the matrix. As with vectors, either the global or local size may be provided, with PETSc automatically computing the other. And again we specify the local sizes to be consistent with the case for when grid partitioning information is provided as part of the mesh file. The variable which defines the matrix, \underline{A} , is `(Mat_A)`.

A more sophisticated matrix creation command has also been implemented in this work using the sparse AIJ format. This matrix creation command improves performance over the basic matrix creation command given above. This is accomplished by a call

```
MatCreateMPIAIJ(PETSC_COMM_WORLD, PETSC_DECIDE,
                PETSC_DECIDE,
                nnodemaxtot, nnodemaxtot, dnz, PETSC_NULL_INTEGER, onz,
                PETSC_NULL_INTEGER, Mat_A, IERR)
```

where the second and third arguments (`PETSC_DECIDE`) indicate that PETSc is to determine the local row and column size of the matrix when it is distributed over multiple processors, while the next two arguments (`nnodemaxtot`, `nnodemaxtot`) indicate the row and column size of the matrix. As these matrices being solved by CHAD are square, the two terms are identical. The number of non-zeroes per diagonal submatrix, and for the off-diagonal matrix, are specified using (`dnz` and `onz`). These terms are memory pre-allocation terms for PETSc and are set to 7 and 6 respectively, using the assumption that grids being solved using PETSc are primarily of hexahedral elements. These settings will overestimate the memory requirements for primarily tetrahedral meshes, and should be adjusted accordingly. Setting these values lower than required can

substantially reduce the performance of PETSc as the process of allocating memory is intrinsically very expensive. The effect on performance of these memory allocation terms is discussed in greater detail in Chapter 6. The `PETSC_NULL_INTEGER` terms are placeholders for an alternate method of specifying the nonzero structure of the matrix and are not used here.

Once the matrix variable (`Mat_A`) has been constructed, the matrix options must be set in a fashion analogous to that of the vector creation and is programmed as

```
MatSetFromOptions(Mat_A, IERR).
```

Once the matrix system has been created, a solver is required to compute the unknown vector. The construction of a linear solver context, (`SLES`), is simple and requires only a creation call

```
SLESCreate(PETSC_COMM_WORLD, sles, IERR).
```

5.3 Loading Data into PETSc

After the vectors, matrix and linear solver context are all constructed, the matrix and residual vector must be filled with data. As each processor *owns* its own data, and the PETSc vector is of a global size (`nnodesmaxtot`), each processor must ensure it loads its corresponding data into the proper global location of the PETSc vector. Each processor therefore loops over all the nodes it owns and assigns the data into a PETSc defined scalar. An integer value, `nloc`, is set to the global node number of each node from the `CHAD` array which defines the global node numbers (`NODENOG`). Together, these two variables are used with the PETSc routine `VecSetValue`

```

do I=1,nnodemax

    TMP = BB(I)

    nloc = NODENOG(I)

    call VecSetValue(PTC_B,nloc1,TMP,INSERT_VALUES,IERR)

end do

```

The routine

```
VecSetValue(PTC_BB, nloc-1, TMP, INSERT_VALUES, IERR)
```

inserts values loaded into the PETSc scalar `TMP` into a vector cache at the global location `nloc-1`. FORTRAN arrays begin by default at index value of 1 while C arrays begin at index value of 0, and thus a minus-one is required at the insertion point. The parameter `INSERT_VALUES` directs PETSc to overwrite any existing values in specified vector position.

Upon completion of the loop, the vector is assembled using the two routines

```
VecAssemblyBegin(PTC_B, IERR)
```

and

```
VecAssemblyEnd(PTC_B, IERR)
```

which moves the values from a cache into the actual vector. It is possible to insert additional code between the calls `VecAssemblyBegin` and `VecAssemblyEnd` in order to maximize processor utilization. For example, it might be possible to improve

computation times by loading the matrix into cache while assembling the vector. This has not been investigated.

The vector `PTC_P` may be preloaded with values that will serve as an initial guess for the solver. One possible guess is to use the estimated change in pressure from the previous iteration. To do this requires the use of an array to store the previous values of the change in pressure. A possible choice for this array would be to use the `DELTAP` array since it is already available. However, this array is *deallocated* during the iteration process and the values would therefore be lost. Therefore, a new array `PDP` was created within the `arrays_glo_module_c` routine. At the end of the continuity solver updates, the computed values of the change in pressure are copied to this array. In section 5.4 we explain how the solver utilizes this initial guess.

Matrix construction occurs in a fashion similar to vector construction, including copying the variable to a scalar and then setting that scalar to a global matrix location.

```

do I=1,nnodemax

    TMP=A0(I)

    nloc = NODENOG(I)

    call MatSetValue(Mat_A,nloc-1,nloc-
        1,TMP,INSERT_VALUES,IERR)

end do

```

Here, the CHAD array A0 contains the diagonal terms for the matrix which are copied to a temporary variable TMP and inserted in the global cache position given by NODENOG using the PETSc routine

```
MatSetValue(Mat_A,nloc-1,nloc-1,TMP,INSERT_VALUES,IERR).
```

As this is the diagonal of the matrix, the PETSc locations are identical for both the row and the column and are given by nloc-1 and nloc-1 respectively.

The off-diagonal terms are also loaded in a similar fashion using

```
do J=1,nconns
    k1 = NODEST(1,J)
    k2 = NODEST(2,J)
    TMP = A0T(1,J)
    MatSetValue(Mat_A,k1-1,k2-1,TMP,INSERT_VALUES,IERR)
    TMP = A0T(2,J)
    MatSetValue(Mat_A,k2-1,k1-1,TMP,INSERT_VALUES,IERR)
end do
```

except in this case the row and column locations are determined from the array NODEST which identifies the nodes residing at the end of each connection (k1 and k2). As node k1 identifies the lower node number on a connection across a cell face (and therefore resides in a cell center), it identifies the row position of that cell center for the matrix coefficient. The neighboring node (k2) is thus across the cell face (which is denoted by the connection) and is therefore the off-diagonal term. It represents the column value of the matrix entry.

The syntax for setting the matrix values into the cache is

```
MatSetValue(Mat_A, k1-1, k2-1, TMP, INSERT_VALUES, IERR)
```

where the row and column locations are respectively represented by $(k1-1)$ and $(k2-1)$ within PETSc. Again, the minus-one is due to the different forms of indexing between C and FORTRAN.

The matrix is assembled by moving values from the cache to the actual matrix using the routines similar to those used in the vector assembly

```
MatAssemblyBegin(Mat_A, MAT_FINAL_ASSEMBLY, IERR)
```

```
MatAssemblyEnd(Mat_A, MAT_FINAL_ASSEMBLY, IERR) .
```

Once again, while it is possible to insert additional code between these two statements to optimize the code, this has not been tried here.

Once the matrix has been assembled, assertions may be made about future constructions of subsequent matrices. In particular, if the mesh connectivity does not change, the non-zero locations of the matrix will remain the same. This performance oriented assumption has been made available in CHAD using the routine

```
MatSetOption(Mat_A, MAT_NO_NEW_NONZERO_LOCATIONS, IERR) .
```

This routine is currently disabled (commented out) in CHAD_UI. This is because at the present time, the matrix system, vectors, and solver context are all destroyed once the solution to the continuity equation is obtained in order to free up memory. If, at a future time it becomes desirable to not release and then reallocate the associated memory, and instead retain it in global memory, then the above subroutine should be used.

The linear solver parameters are set using the routine

```
SLESetOperators(sles, Mat_A, Mat_A, SAME_NONZERO_PATTERN,  
               IERR)
```

where the second argument (`Mat_A`) indicates the matrix being used and the second variable (`Mat_A`) indicates the matrix to be used to construct the preconditioner for the system. In this case, the coefficient matrix is the same matrix that is used to construct the preconditioning matrix. The `SAME_NONZERO_PATTERN` indicates information about the preconditioner, in this case that the non-zero pattern remains the same.

The options for the linear solver are set using the routine

```
SLESetFromOptions(sles, IERR)
```

in the same way as was done with the other PETSc constructs.

5.4 Solving Matrix Systems and Extracting Results

The Krylov Space controls are accessible through

```
SLEGetKSP(sles, ksp, IERR).
```

Tolerances that control convergence are accessible through

```
KSPGetTolerances(ksp, rtol, atol, dtol, max_iterations)
```

and a similar routine for setting tolerances

```
KSPSetTolerances(ksp, rtol, atol, dtol, max_its).
```

In our implementation of PETSc, `rtol` (relative tolerance), `atol` (absolute tolerance) and `dtol` (divergence tolerance) retain their default values (10^{-5} , 10^{-50} , 10^5), but we set

the maximum number of iterations to the CHAD restricted limit that it would have used for the GCR solver. The PETSc default for `max_its` is 10^5 , which greatly exceeds CHAD's hardcoded max of 300. We recognize that the CHAD limit may not be suitable for the PETSc solver, however, as will be described further in Chapter 6, the solution controls that permit time-stepping in CHAD are tailored with a slew of empirical constants that are apparently tuned to the GCR solver for both outer and inner iteration optimization. In summary, the advancement of time-stepping is very sensitive to the number of iterations required to solve the continuity equation during the *first* outer iteration. With PETSc, the number of iterations required to obtain convergence will principally be controlled through `rtol` which can only be approximately guessed at runtime. As such we have introduced a very simple method for setting `rtol`, and it involves providing the value of `rtol` to `KSPSetTolerances`. The new value of `rtol` is stored in a new variable `petsc_rtol` when another new variable, `float_rtol`, is found to be true in the input file.

If the user wishes to specify a non-zero initial guess, the routine

```
KSPSetInitialGuessNonzero(ksp,PETSC_TRUE,IERR)
```

should be called. `PETSC_TRUE` should be changed to `PETSC_FALSE` when an initial guess is not desired. Alternatively, the line may be commented out. When `PETSC_FALSE` is specified, the initial guess is explicitly set to zero. However, as the solution is already set to zero in PETSc when no initial guess is specified, no additional work is explicitly saved.

The linear system is solved by calling

```
SLESSolve(sles, PTC_B, PTC_P, ITS, IERR)
```

where the solution vector (PTC_P) is returned in PETSc form within PTC_P, and ITS is an integer number indicating the number of iterations that were required to obtain convergence.

Convergence is obtained in SLESSolve when

$$\|\underline{r}_k\|_2 < \mathbf{max}(\mathit{rtol} * \|\underline{b}\|_2, \mathit{atol}) \quad (5.1)$$

where

$$\underline{r}_k = \underline{b} - \underline{A}x_k. \quad (5.2)$$

The default for *rtol* is 10^{-5} while the default value for *atol* is 10^{-50} . These tolerances may be overridden when executing CHAD using the command line options `-ksp_rtol (rtol)` and `ksp_atol (atol)`.

Just as data must be inserted into a PETSc usable form via subroutine calls to `VecSetValue` and `MatSetValue`, data must be extracted from the PETSc form for use within CHAD. The procedure `VecGetArray` is used to access solution results usable by CHAD. As not all features available in C are found in FORTRAN, and since not all FORTRAN 90 compilers are supported for this particular feature of PETSc, a special FORTRAN 77 syntax is required. The call essentially provides a 1-D array to PETSc which uses a second integer index in the procedure call to access PETSc's internal array storage. The syntax within CHAD is written as

```
VecGetArray(PTC_P, P_ARRAY, I_P, IERR)
```

where `P_ARRAY` is the 1-D vector, and an integer offset which provides the appropriate location within PETSc's internal storage system is given by `(I_P)`. The data can then be extracted and assigned to a FORTRAN array `DP` with the following code:

```
do I=1,nnodemax
    DP(I)=P_ARRAY(I_P+I)+DP(I)
end do
```

The data returned by `VecGetArray` are local and based on the assumption that the local processor vector sizes match between PETSc and CHAD. When node partitioning information is provided as part of the meshfile, local vector and matrix sizes must be provided. If global sizes are provided to PETSc when custom grid partitioning is used, it is possible (and probable) that the expected node distribution will be different than how PETSc would distribute them across processors. For example, imagine a 100 node problem to be run on two processors. If CHAD partitions the problem on its own, it will internally distribute nodes 1-50 on processor one, and nodes 51-100 on processor two. Similarly, when PETSc creates a vector it will distribute nodes 1-50 on processor one and nodes 51-100 on processor two. Then, when `VecGetArray` is called, the processors will retrieve nodes 1-50 and 51-100 respectively from PETSc vectors of the same size. The extraction process will therefore have functioned as intended. However, in a problem where the user specifies that nodes 1-30 belong on processor number one and 31-100 belong on processor number two, PETSc will still attempt to distribute the nodes as 1-50 and 51-100 if global sizes are supplied. Thus when `VecGetArray` is used, processor number one will retrieve 1-30 and processor number two will retrieve 31-

100, except that there exists only 50 values in the PETSc solution vector, so the last 20 values of the first PETSc vector solution are lost and all the values retrieved for the second vector will be incorrect.

5.5 Terminating PETSc

After extracting the solution vector (PTC_P) into a CHAD usable form (DP), a final call to PETSc must be made to restore the array using the call

```
VecRestoreArray(PTC_P, P_ARRAY, I_P, IERR) .
```

The PETSc manual makes an extra effort to remind the user of the importance of this call.

Once convergence is achieved, the memory allocated to PETSc should be released. This is accomplished by calling the proper PETSc routines as is shown below:

```
VecDestroy(PTC_P, IERR)
```

```
VecDestroy(PTC_B, IERR)
```

```
MatDestroy(PTC_A, IERR)
```

```
SLESDestroy(SLES, IERR)
```

It may be desirable to create and destroy the PETSc data structures only once within the flow of the program instead of once per outer iteration. If the entire program resides within memory it would likely be desirable to create the structures once and destroy them at the end, reusing them when applicable. However, for the purposes of this work, they are destroyed to free memory for the rest of the system. And in fact, as will

be described in Chapter 6, the most expensive aspects of PETSc when used with CHAD are not related to memory allocation and cleanup activities.

At the end of a simulation, PETSc must be explicitly terminated. This occurs in `chad_main` with a call to `petsc_chad_finalize`. This routine has the syntax

```
petscfinalize(PETSC_NULL_CHARACTER, IERR)
```

where the arguments are exactly analogous to those found in the initialization procedure.

Table 5.3 summarizes the calls directly required for the PETSc implementation, including their location. The order of the calls in Table 5.3 corresponds to the order they are executed within CHAD, so it is possible to discern the sequence of PETSc events directly from this table.

Table 5.3: PETSc Calls in CHAD for Parallel Computations

Call	Function	Location
Petsc_initialize	initializes PETSc	petsc_chad_init
Solve_petsc_cont_par	call the parallel PETSc solver	solve_or_res_nl_cont
VecCreateMPI	create parallel PETSc vector	solve_cont_petsc_par
VecSetFromOptions	set vector parameters	solve_cont_petsc_par
VecDuplicate	create duplicate PETSc vector	solve_cont_petsc_par
MatCreateMPIAIJ	create parallel AIJ PETSc matrix	solve_cont_petsc_par
MatSetFromOptions	set matrix parameters	solve_cont_petsc_par
SLESCreate	create PETSc solver	solve_cont_petsc_par
VecSetValue	set vector entries	solve_cont_petsc_par
VecAssemblyBegin	begin to loading entries from cache to actual vector	solve_cont_petsc_par
VecAssemblyEnd	finish loading entries from cache to actual vector	solve_cont_petsc_par
MatSetValue	set matrix entries	solve_cont_petsc_par
MatSetOption	set new matrix option	solve_cont_petsc_par
SLESSetOperators	set solver operators	solve_cont_petsc_par
SLESSolve	Solve $Ax=b$	solve_cont_petsc_par
VecGetArray	retrieve b from PETSc	solve_cont_petsc_par
VecGetOwnershipRange	retrieve range of nodes owned by this processor	solve_cont_petsc_par
VecRestoreArray	release PETSc array access	solve_cont_petsc_par
VecDestroy	release vector memory	solve_cont_petsc_par
MatDestroy	release matrix memory	solve_cont_petsc_par
SLESDestroy	release solver memory	solve_cont_petsc_par
petsc_finalize	set matrix parameters	petsc_chad_finalize

Chapter 6: Testing the PETSc Implementation

6.1 Testing Overview

The CHAD 5.0 package includes eleven test problems. These problems may be used to benchmark CHAD installation, test changes users make to CHAD, or perhaps even to familiarize oneself with CHAD. The test problems (hereafter referred to as sample problems) span a range of different physics models and code capabilities, such as distorted meshes, severe tests of advection schemes, engine compression and expansion, transient thermal diffusion and shock waves.

The ‘reference’ solutions to the eleven sample problems against which CHAD’s results may be compared can be obtained from a variety of sources. Some of the sample problems have analytical solutions. Other ‘reference’ (benchmark) solutions are obtained using fine-mesh numerical schemes, and may be found in the literature. For some problems, the ‘reference’ solution is given in CHAD’s documentation about the test problems.

One of the sample problems was selected to test the reformulated continuity equation, described in chapter three, and the PETSc implementation, described in chapters four and five. This widely used lid-driven cavity flow problem, has benchmark solutions for a variety of Reynolds numbers (Ghia et al., 1982.)

Ghia's benchmark solutions for lid-driven cavity flows at different Reynolds numbers were obtained using multigrid methods and an incompressible fluid. Tabulated results are available for select variables at certain locations in the cavity. Plotting these variables along a line provides a way to graphically compare the benchmark solutions against user obtained solutions.

CHAD's sample lid-driven cavity problem consists of a mesh file containing the information for a $30 \times 30 \times 1$ cell grid and an input file which specifies general problem settings, such as hydrodynamic mode (explicit, semi-implicit, implicit etc), time step sizes, material properties and the equation of state to be used. Notably, the input file specifies the use of a perfect equation of state, which is compressible. This deviation from the benchmark problem is not significant as the flow in the lid-driven cavity problem is nearly incompressible. Generally, compressible fluids may be considered incompressible when Mach number is less than 0.2 (or, the fluid speed is less than 20% of the speed of sound in the fluid).

In addition to the mesh file provided with the CHAD distribution, mesh files were generated to study the effect of mesh refinement on accuracy. These new mesh files were created using in-house mesh-generation capabilities. Meshes of $60 \times 60 \times 1$, and $120 \times 120 \times 1$ cells were developed.

Flows with different Reynolds numbers were simulated by modifying the kinematic viscosity of the fluid in the input file. Maximum and minimum allowable time

step sizes (DTMAX, DTMIN) were also changed to agree with the stability conditions associated with different grid densities. Use of the PETSc implementation and the reformulated continuity equation were specified by adding flags to the input file as described in the previous chapters.

Before presenting results for the benchmark problem, the computing environment on which the CHAD simulations were conducted is described. While the computing environment will have little effect on the accuracy of our results, the solution times required to obtain those results are affected. Additionally, several features, which were added to CHAD to provide information about the solution quality above and beyond those currently available in CHAD, are discussed. These features greatly assist in the analysis of large amount of data generated when solving problems using CHAD. Results for the benchmark problem obtained using CHAD are then presented for single processor simulations of the reformulated continuity equation and the first order scheme that it was based upon. Next, single processor results obtained using PETSc coupled with CHAD's original continuity equation are presented.

After discussing the results obtained by using the reformulated continuity equation and those obtained using PETSc with the original continuity equation, optimization of PETSc's solver performance using various computational controls available in PETSc is discussed. These results were obtained using single processor simulations. Next, performance timings for various PETSc solver and preconditioner

configurations are given. Finally, CPU times for various serial and parallel simulations for different configurations are tabulated. These results are interpreted and discussed.

In the last part of this chapter, results obtained using fixed time step sizes are extended to variable time steps, and results obtained using PETSc and variable time step are compared with those obtained using fixed time step.

6.2. The Computing Environment

The *Reserv* Linux cluster at Argonne's Reactor Engineering Group was used to obtain the solutions to all the problems described in this dissertation. This cluster contains approximately 80 3.4 GHz processors, each with 2 GB of memory, connected by Gigabit networking and running Red Hat Linux 7.2. MPICH is used to connect the machines in a parallel software environment. For technical reasons, it was required to compile a version of MPICH using gcc (GNU Compiler Collection) that is separate from the default MPICH implementation on the *Reserv* cluster. (Due to fear of conflict with other existing software running on *Reserv*, system administration of *Reserv* did not wish to install the MPICH library under the *root* directory. Hence, gcc compiled MPICH was compiled in and runs from the author's home directory.) This provides compatibility between CHAD, PETSc, PGSLib, and several other software packages that was not available using the cluster's default implementation of MPI.

CHAD was compiled on *Reserv* using Absoft's FORTRAN compilers version 8.0 using the second level of block optimizations, specified by using the compiler flag `-O2`.

PGSLib was built using a combination of *gcc* and Absoft's FORTRAN F77 and F90 compilers. The linear algebra packages BLAS and LAPACK were compiled using a combination of the Absoft's compilers and *gcc* compilers. PETSc version 2.1.6 was compiled using the `linux_absoft make` configuration script, Absoft's FORTRAN compilers and *gcc* compilers.

The linking of all the above mentioned software packages for a parallel compilation of CHAD proved to be an extremely formidable task requiring substantial technical expertise. The complicated nature of configuring all the required software means that ANL's *Reserv* cluster is the only location CHAD has successfully been compiled (in parallel) for this dissertation work. Because of the complexity involved, particulars regarding the software installation and configurations used at ANL are described thoroughly in Appendix D. [An effort was in fact made to configure and link all the software packages on the UIUC's Computational Science and Engineering *Turing* cluster. However, despite help from the system administrators recent versions of PETSc could not be properly configured on the *Turing* cluster, while compilations of PGSLib proved troublesome on one of the two networking options available.]

6.3 Solution Monitoring and Convergence Controls in CHAD

CHAD offers very little in the way of convenience for solution monitoring and control. The extent of CHAD's solution monitoring is limited to its output file which displays extreme (min/max) values for variables, iteration counts for the hydrodynamic equations being solved, and some other basic information such as mass, momentum, and

energy totals. One useful output is the number of the node that is limiting convergence. The problem time, whether the timestep size is increasing or decreasing, and the number of time steps taken (cycle) are also output.. Sample output from a typical CHAD simulation is provided below.

```
ncyc      = 0; time      = 0.000000E+00; dt      = 1.000000E-01
crank     = 0.000000E+00; wpiston = 0.000000E+00
avp       = 8.611558E+07; pgs      = 1.000000E+00
iddt      = Grow; dtiter   = 1.999999E-01
dtacc     = 8.988466+207; nnodeacc= 0
dtburn    = 1.797693+208; nnodebrn= 0
dtcour    = 1.348270+208; nconncou= 0
dtdiff    = 8.988466+207; nnodedif= 0
dtstrain  = 1.000000E+07; nnodestr= 0
dtmesh    = 1.797693+208
nres_uvw  = 0; nres_tmp= 0; nres_con= 0
```

Hydro Outer-Iteration Data:

```
Out.#= 1; #strs= 0; #uvw = 7; #temp= 5; #cont= 39
Out.#= 2; #strs= 0; #uvw = 11; #temp= 1; #cont= 45
Out.#= 3; #strs= 0; #uvw = 13; #temp= 1; #cont= 50
Out.#= 4; #strs= 0; #uvw = 13; #temp= 1; #cont= 50
```

System Extrema:

```
Pmin      = 8.611558E+07; RHomin = 1.000000E+03; Tmin      = 3.000000E+02
Pmax      = 8.611558E+07; RHOMax = 1.000000E+03; Tmax      = 3.000000E+02
Xmin      = 0.000000E+00; Ymin   = 0.000000E+00; Zmin      = 0.000000E+00
Xmax      = 3.000000E-01; Ymax   = 3.000000E-01; Zmax      = 1.000000E-02
Umin      = 0.000000E+00; Vmin   = 0.000000E+00; Wmin      = 0.000000E+00
Umax      = 0.000000E+00; Vmax   = 0.000000E+00; Wmax      = 0.000000E+00
TKEmin    = 0.000000E+00; EPSmin = 0.000000E+00
TKEmax    = 0.000000E+00; EPSmax = 0.000000E+00
```

System Totals:

```
Mass      = 8.999999E-01; Volume = 8.999999E-04
I         = 1.942275E+05; KE      = 0.000000E+00; E         = 1.942275E+05
Mom. X= 0.000000E+00; Mom. Y= 0.000000E+00; Mom. Z= 0.000000E+00
Ang.M. X= 0.000000E+00; Ang.M. Y= 0.000000E+00; Ang.M. Z= 0.000000E+00
Vort. X= 0.000000E+00; Vort. Y= 0.000000E+00; Vort. Z= 0.000000E+00
TKE       = 0.000000E+00; EPS    = 0.000000E+00; HeatLoss= 0.000000E+00.
```

This data, which must be redirected to an output file, is an ineffective means of interactively determining solution progress and quality. In terms of solution progress, only the current solution time, time step size, and the limiting factor for time step controls are useful. No information is provided to assess overall solution quality, while only

minimal information can be obtained through system totals and minimum and maximum variable values.

Convergence controls within CHAD are similarly limited. Virtually all CFD codes utilize criteria which are used to determine when to halt code execution. Typically, CFD codes are halted at some user specified criteria such as maximum cpu or wall times, or when the computed solution converges to within some tolerance as indicated by the residual for each balance equation. In CHAD, code execution is halted when any one of six conditions are met. They are:

1. If the time step (`ncyc`) exceeds the maximum number allowed (`maxcyc`)
2. If the crank angle (`crank`) exceeds the maximum allowed (`camax`)
3. If the problem time (`time`) reached exceeds the maximum allowed (`tmax`)
4. If it is not converged (`converged`), meaning time step advancement is not possible
5. If there is a problem computing the timestep, resulting in a `timestep_error`
6. If the maximum cpu time is exceeded (`cpu_max`)

Of these six tests, only tests four and five relate to the progress of the solution algorithm in obtaining solutions to the governing equations, and even then they only indicate a *failure* of the solution process. The other four conditions merely specify a

maximum amount of time that the problem is permitted to run. None of these conditions actually suggest convergence has been achieved.

Three new tools were added to CHAD to assist in solution monitoring and control. The first tool allows *monitoring* of variables of interest at selected points. By selecting a location of interest within a problem domain, and variables to be monitored at that location, convergence can be inferred. [A converged solution to a steady state problem is a solution that should not change as iterations advance.] A second convergence monitoring tool that has been added to CHAD measures the overall decrease in variable residuals. Unless a problem begins a computation with a solution that nearly matches a converged solution, a decrease in the residuals of the variables should be observed. A third tool that has been added to CHAD is the ability to halt code execution based upon a predetermined tolerance in variable residuals. When the residuals in these variables have been reduced to below a predefined tolerance, the code will halt execution. Each of these additions to CHAD is described in more detail below.

6.3.1 Monitoring Points

The option of monitoring the value of selected variables at selected nodes during code execution has been added into CHAD. This feature may be activated by setting a logical flag that controls monitoring point controls (`plot_mon`) to `true (.true.)` within the input file used by CHAD. When this flag is set to `.true.`, a file by the name of `'monitors.dat'` is created in the directory from where the code is executed. The value of a variable associated with a (global) node number specified by keywords within

the input file is written in `monitors.dat`. Table 6.1 defines the keywords that may be specified to select u , v , and w components of velocity as well as temperature, pressure, turbulent kinetic energy and turbulent dissipation rate. Time is written for all cases. Currently, only one monitoring point per variable may be selected, though different nodes may be selected for different variables. A sample problem that is described more thoroughly later in this chapter illustrates the use of monitoring points for two different numerical schemes denoted as A and B. The problem is that of driven flow in a two-dimensional cavity of 0.3 m x 0.3 m. A monitoring point was selected at $x = 0.23$ m and $y = 0.23$ m, which places it between a large vortex and a region of high velocity that results from the deflection of the fluid near a moving lid against the side of the cavity. Figures 6.1a and b show the nodal u and v velocities at this location plotted against time.

Table 6.1: Variables Associated with Monitoring Points

Variable	Type	Description
<code>plot_mon</code>	logical	activates monitoring point output
<code>imon_node_u</code>	int_kind	global node number for monitoring u component of velocity
<code>imon_node_v</code>	int_kind	global node number for monitoring v component of velocity
<code>imon_node_w</code>	int_kind	global node number for monitoring w component of velocity
<code>imon_node_t</code>	int_kind	global node number for monitoring temperature
<code>imon_node_p</code>	int_kind	global node number for monitoring pressure
<code>imon_node_k</code>	int_kind	global node number for monitoring turbulent kinetic energy
<code>imon_node_e</code>	int_kind	global node number for monitoring turbulent dissipation

As time progresses, the value of these velocities (as well as other variables not plotted) should approach a constant value. For steady flow, constant values are an indication of

convergence. As seen in the figures, the velocities for scheme B have approached a constant value, while those from scheme A are still showing changes with time. In this particular case, the computation was stopped by condition three mentioned above, that is the problem time reached a maximum allowable time.

6.3.2 Residual Reduction Monitoring

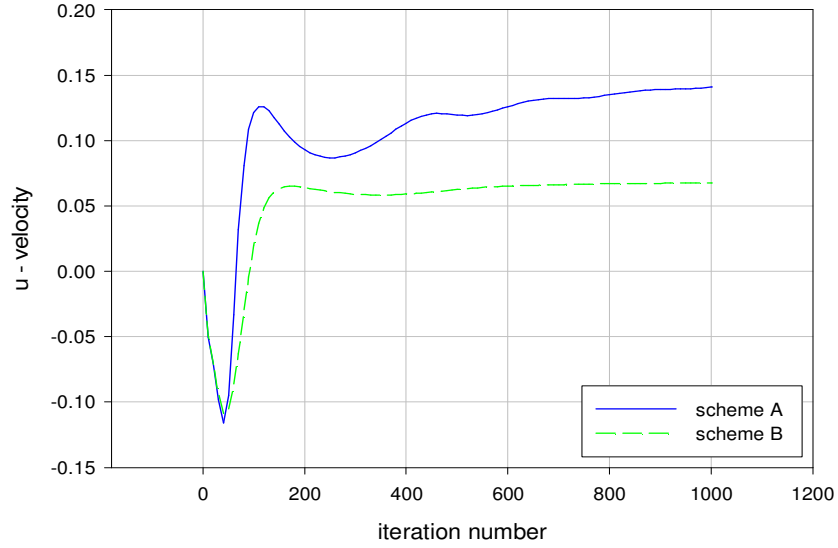
Another method for determining convergence is based upon the residual associated with each of the system variables. In general, residuals associated with the continuity, momentum and energy equations (as well as turbulence model variables) are monitored and a drop to a pre-specified level is an indication of convergence. [In cases where the initial guess for variables very nearly matches the solution, the residuals are already low at the beginning of iterations.] A tool for monitoring the residuals has been developed for CHAD. It may be enabled by setting the variable which controls residual monitoring (`plot_res`) to true (`.true.`) within the input file.

Residuals are monitored using *residual reduction factors*. The residual reduction factors for the velocity components are given by

$$r_{red,q} = \frac{\sum_v |(R_q)_v|}{\sum_v \sqrt{u^2 + v^2 + w^2}} \quad (6.1)$$

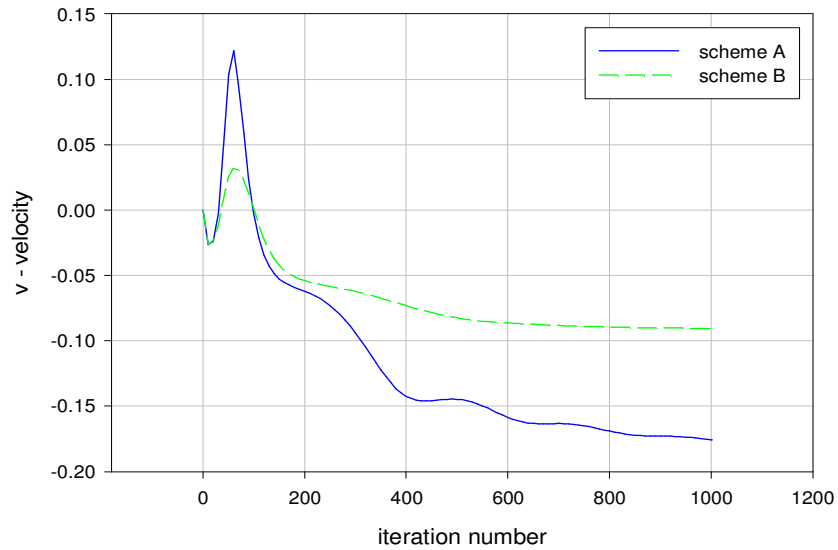
where q represents a particular component of velocity. The residual reduction factor is computed at the end of the momentum computations in the routine `solve_or_res_nl_cont_uvw`.

Monitoring Point U - Velocity for Two Schemes
at $X=0.23\text{m}$, $Y=0.23\text{m}$



(a)

Monitoring Point V - Velocity for Two Schemes
at $X=0.23\text{m}$, $Y=0.23\text{m}$



(b)

Figures 6.1: u and v velocities at a particular point in the domain in an example problem obtained using two different numerical schemes. After 1000 steps, Scheme A is still not converged, while scheme B is nearly converged.

Similar reduction factors are defined for temperature and continuity equations, though with the continuity equation there is one difference—the normalization term is computed as the maximum value of the first ten iterations.

$$r_{red,p} = \frac{\sum_v |(R_p)_v|}{\max(|R_{p,v}^1|, |R_{p,v}^2|, \dots, |R_{p,v}^{10}|)} \quad (6.2)$$

The variables used to implement the residual reduction monitoring are tabulated in Table 6.2. Enabling the residual monitoring feature causes CHAD to create a file called `residuals.dat` in which the time and the residual reduction factors for u , v , w , t , and p are written at every time step. Residual reduction factors for turbulent kinetic energy and dissipation rate have also been programmed into CHAD, but currently are not written in the output file.

The sample problem described earlier for monitoring points is used here to illustrate the usage of the residual reduction factors. Figure 6.2 shows the residual reduction factor computed for the same sample problem described earlier. Here, the u and v velocity residuals approach an oscillatory pattern while pressure gradually decreases. Residual reduction factors for all variables drop by about one order of magnitude before the simulation was stopped. As the residuals decrease, the solution is expected to be converging. The level to which the residuals should drop for satisfactory convergence is typically left to the user and depends at least in part on the nature of the problem being solved.

Table 6.2 Variables Associated with Monitoring Points

Variable	Type	Description
plot_res	logical	activates residual reduction monitoring output
norm_fact_u	real_kind	normalization factor for u-velocity
norm_fact_v	real_kind	normalization factor for v-velocity
norm_fact_w	real_kind	normalization factor for w-velocity
norm_fact_t	real_kind	normalization factor for temperature
norm_fact_p	real_kind	normalization factor for pressure
r_red_fact_u	real_kind	residual reduction factor for u-velocity
r_red_fact_v	real_kind	residual reduction factor for v-velocity
r_red_fact_w	real_kind	residual reduction factor for w-velocity
r_red_fact_t	real_kind	residual reduction factor for temperature
r_red_fact_p	real_kind	residual reduction factor for pressure

6.3.3 Computation Stopping Criteria

As was mentioned earlier, CHAD does not stop computations based upon any measure of convergence, excluding divergence. To add some user control that dictates when computations should stop, the residual reduction values described in the previous section were used to construct a seventh user-defined control parameter.

Several new variables were added to CHAD to implement this new control. These values, once read from the input file, provide convergence targets for particular residuals. When all of the residual reduction factors are reduced to below the specified convergence

limits, which are specified by the user in the input file, the solution may be considered converged according to this criteria, and code execution is halted.

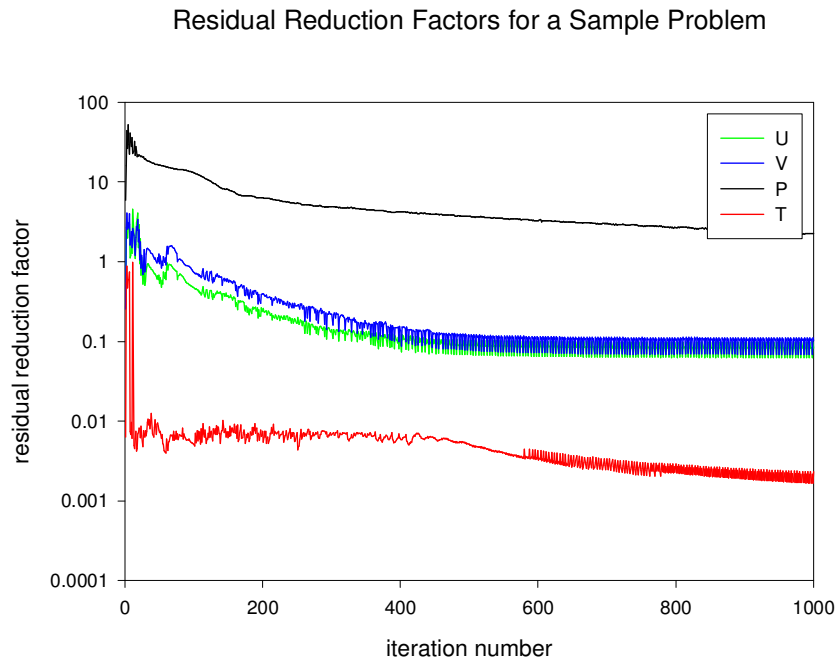


Figure 6.2: Residual reduction factor of velocities, temperature, and pressure for a sample problem. This problem, which uses a coarse grid and lax convergence criteria, results in a decrease of less than two orders of magnitude in the residual reduction factors before oscillations develop.

Many CFD codes use residual reduction values as a primary indicator of convergence, often using a value of 10^{-3} as the default threshold which suggests satisfactory convergence. In this implementation of CHAD, the residual reduction tolerances are set by default to `tinynum` which is on the order of the smallest exponent available on a given machine. On the *Reserv* cluster, `tinynum` is approximately $2.2\text{E}-208$. Currently, all convergence targets must be specified in the input file if the computation is to halt based upon this method of control (as the targets of `tinynum`

would otherwise never be reached). Table 6.3 describes the new variables used in this control.

Table 6.3 Variables Used for Convergence Targets

Variable	Type	Description
conv_res_u	real_kind	residual convergence target for u component of velocity
conv_res_v	real_kind	residual convergence target for v component of velocity
conv_res_w	real_kind	residual convergence target for w component of velocity
conv_res_t	real_kind	residual convergence target for temperature
conv_res_p	real_kind	residual convergence target for pressure

6.4 Solutions to the Lid-Driven Cavity Problem

The two-dimensional lid-driven cavity problem has a well accepted benchmark solution (Ghia et al., 1982). It is a difficult test problem for CFD codes, and happens to be one of the sample problems provided with the CHAD distribution package. For these reasons it is used here to test the various formulations of the continuity equation and the PETSc implementation discussed in the previous chapters.

Though the lid-driven cavity problem is traditionally solved with incompressible fluids and the benchmark solutions are also obtained for the incompressible case, CHAD does not have an incompressible equation of state. However, the problem is expected to be nearly incompressible, and solutions to this problem computed by CHAD actually show very limited compressibility effects ($< 0.01\%$), which is expected since fluid velocities are well below the speed of sound. Therefore, an almost incompressible fluid

flow problem is being solved by CHAD and hence the results are expected to be similar to the benchmark solutions. However, we note at the outset that an increased computation time will be required due to the computation of the equation of state and changes in density that result from changes in pressure.

We stop to note that later in this dissertation we describe the addition of an incompressible equation of state to CHAD. However, in this chapter the focus is comparing solutions obtained using different options in CHAD against each other and not against the benchmark solutions, that is, the solutions obtained using the reformulated continuity equation and using PETSc are compared against solutions obtained using version of CHAD 5.0 without these features. The benchmark results, however, do serve as a point of reference, indicating whether our solutions are better or worse than those obtained using CHAD 5.0 without PETSc or the reformulated continuity equation. When the incompressible equation of state is discussed in chapter seven, results will be more directly compared against the benchmark solutions.

6.4.1 Lid-Driven Cavity Model Description

The input file and mesh file of the basic lid-driven cavity problem are provided with the CHAD distribution. The (X, Y, Z) grid is 31 x 31 x 2 nodes with the Z-direction artificially treated as a symmetry plane. However, we solve the problem in 3-D mode due to concerns with the way symmetry boundary conditions are treated in CHAD. The problem dimensions are 0.3 m x 0.3 m x 0.001 m. It is shown with a simplified grid (7 x 7 x 1) in Figure 6.3 where ‘omitted’ nodes are assumed to be ‘standard’ interior nodes in

the X-Y plane having no special relevance to the problem description. Velocity is prescribed on the nodes on the lid of the cavity, while no-slip boundary conditions are imposed on the left, right and bottom walls of the domain. The top left and right corners of the problem are no-slip nodes superseding the prescribed velocity. All interior nodes are free-slip in type. CHAD uses a 3 x 3 symmetric constraint tensor written in an array (table) called CONTAB (constraint table) to modify velocity boundary conditions. This array plays an important role in CHAD, and is frequently mentioned in latter parts of this dissertation. A brief description of this array and its role in CHAD is given in Appendix C.

Free-slip nodes in CHAD are treated as interior nodes, though constraints on velocity (and indirectly other boundary conditions) may be applied to them depending on the corresponding value in the CONTAB array. In a symmetric problem, velocity constraints are applied in one dimension to prevent flow in that direction.

At walls, free-slip nodes may be used when the boundary layer at the wall is completely resolved, or when information at the wall is not required. When the boundary layer at the wall is not completely resolved as is typical for turbulent flow, no-slip nodes are used. In this case, variables at the wall are obtained using wall functions. In both cases, CONTAB arrays are used to apply constraints on velocity as follows:

1. For nodes on all external boundaries, including lid – all corresponding values in the CONTAB array are zero.
2. On all internal nodes, the constraint table is given as CONTAB(1,0,0,1,0,0) which permits freedom in the XY directions, but not the Z direction.

Lid velocity is specified with velocity source term variables defined in the input file as USRC, VSRC and WSRC. As the lid velocity in this problem is in the X direction, VSRC and WSRC are both zero while USRC is set to 1.0 m/s. The Reynolds number for the problem is 1000, which is achieved by setting density to 1 kg/m³ and kinematic viscosity to 0.0003 m²/s. The Reynolds number for different problems is changed by adjusting the kinematic viscosity (and not the lid velocity or geometry dimensions). Temperature is specified as 300 K.

As CHAD 5.0 has no mechanism to stop code execution upon satisfaction of a convergence criterion, the cavity problem was run from $t = 0.0$ to $t = 10$ seconds, which was the time limit specified with the input file provided with the CHAD distribution package. The number of time steps required to achieve a time of $t = 10$ seconds varied between implementations used, with the reformulated continuity equation, and the PETSc formulation based upon the reformulated continuity equation requiring smaller time step sizes in order to maintain stability.

The results for five different versions of CHAD are presented along with the benchmark results (Ghia et al., 1982). The first set of results represents CHAD without any changes—which we will refer to as original, or *ori* CHAD. Additionally, CHAD was run using the first order upwinding (FOU) scheme for velocity, density, and enthalpy and this will be referred to as *upw*. Then results will be presented for the reformulated continuity equation. This is built upon the FOU scheme and this shall be referred to as *anl*. Finally, results obtained using PETSc solvers will be referred to as *ptc*.

As the FOU scheme is not as accurate as CHAD’s second order scheme, we expect the results to be poorer. In particular, results are expected to show numerical diffusion, which is a characteristic of FOU schemes. Additionally, since the *anl* and *ptc* schemes are built upon the FOU scheme, we also expect those schemes to perform similar to the FOU scheme. Later in the chapter, we present results obtained using PETSc with CHAD’s native continuity equation. We will refer to that as *ptc-o*.

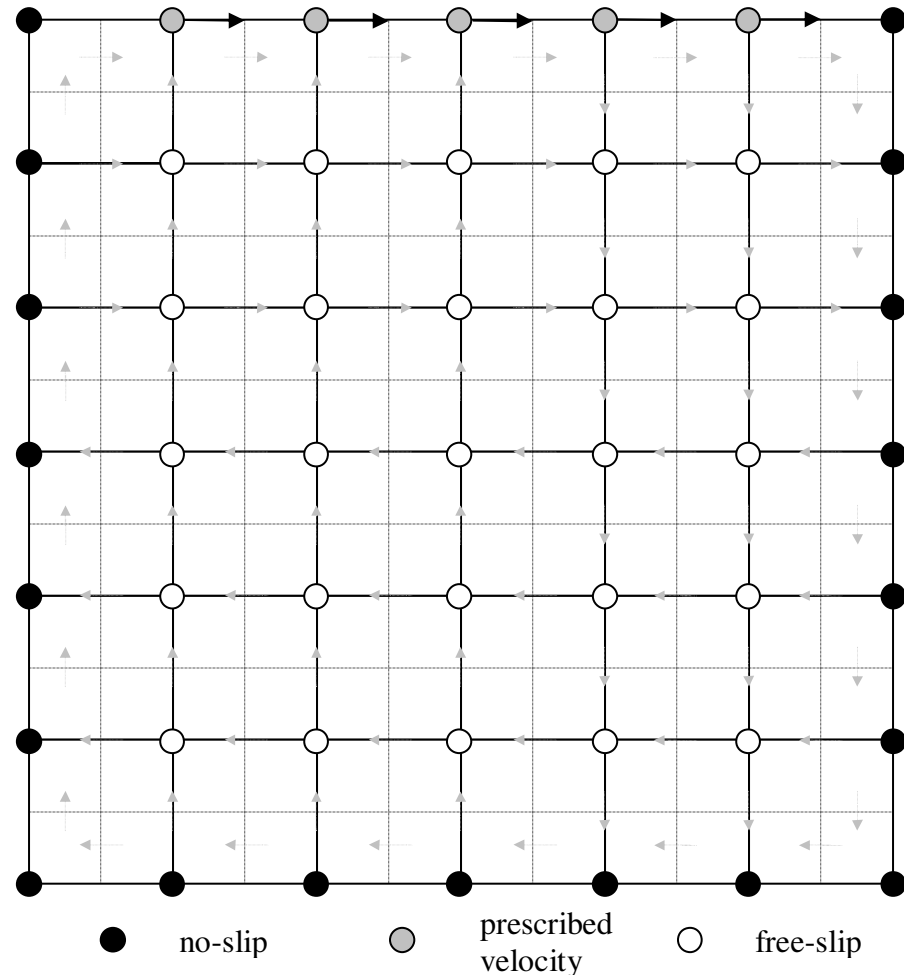


Figure 6.3: Simplified Lid-Driven Cavity Mesh. A 1-D mesh of $7 \times 7 \times 1$ is depicted instead of the full $31 \times 31 \times 2$. No-slip nodes are in black, prescribed velocity nodes in grey and free-slip nodes in white. Gray arrows indicate cell-face-fluxes.

6.4.2. Lid-Driven Cavity Results with the First-Order Schemes and CHAD/PETSc

The results obtained using original CHAD (*ori*) and CHAD modified to use the first order schemes (*upw*, *anl*, *ptc*) are presented first. For purposes of clarity, the results

obtained using CHAD and PETSc based upon CHAD's native continuity equation are presented later.

The cavity flow problem was solved for Reynolds numbers of 100 and 1000. In each case the v -velocity across a line that bisects the left and the right side of the cavity into two equal sized parts (Figure 6.4) is compared against benchmark results. The u -velocity across a line bisecting the top and the bottom of the cavity into two equal sized parts is also compared against the benchmark results.

Figures 6.5 shows the u and v velocity for the $Re = 100$ case on a 30×30 grid (31×31 nodes). The u velocity is captured very well by all four schemes for this relatively simple problem. The v velocity is captured very well, very nearly matching the benchmark results obtained by Ghia et al. with a slight overestimation of the peak, by the second order accurate original version of CHAD.

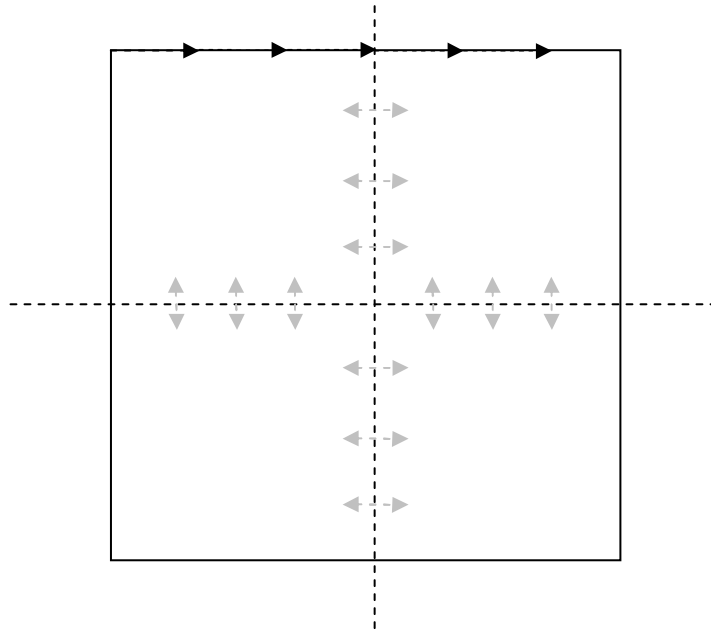


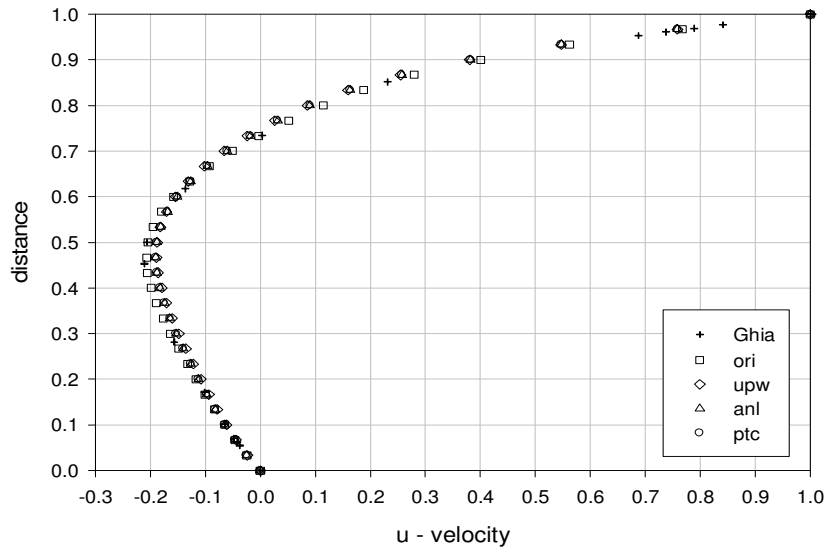
Figure 6.4: Lid-driven cavity results are obtained along two lines which bisect the domain. V-velocity data is presented across the horizontal line while u-velocity data is presented across the vertical line. Gray arrows indicate the component of flow being measured across the lines. Black arrows indicate the direction of the lid velocity.

Solutions obtained using the *upw*, *anl*, and *ptc* formulations also lead to good agreement but are relatively less accurate than the *ori* formulation, underestimating the peak values. This is expected due to the rather diffused nature of the first order upwinding schemes.

Figure 6.6 shows the u and v components of velocity for the $Re = 100$ case on a 60×60 grid. Here, as expected, all schemes show improvement due to the refined grid and better match the benchmark solutions. And again, the original version of CHAD performs well, very nearly matching the benchmark results. The *upw*, *anl*, and *ptc* formulations all perform somewhat worse than the original CHAD, underestimating the peak in the v -velocity.

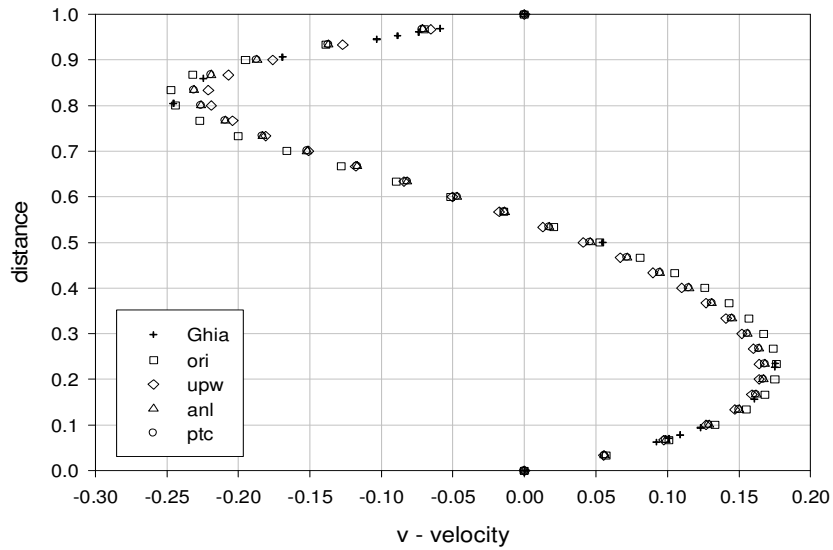
The effect of the diffusive nature of the upwind scheme is more noticeable in the $Re = 1000$ case. The results for the u and v velocities are shown in Figure 6.7 for a 30×30 grid and in Figure 6.8 for a 60×60 grid. Overall, the results are again reasonable considering the diffusive nature of the scheme, though it is apparent that the scheme is not able to adequately resolve the v velocity peaks, even with the higher grid density. However, comparison of these results against results (not shown here) obtained using another FOU scheme (Norris, 2000) shows good agreement, indicating that these results are not unexpected.

u -velocity along Vertical Line Through Center of Cavity
 $Re = 100$ on a 30×30 Grid



(a)

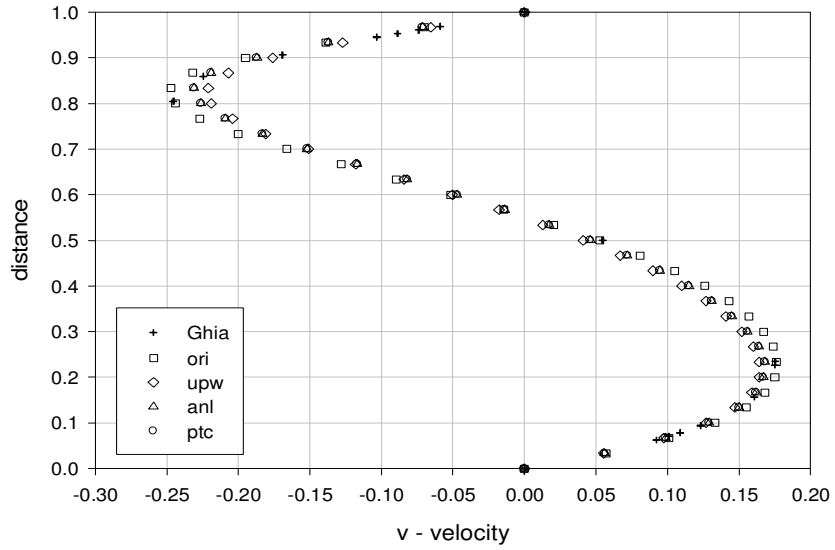
v -velocity along Horizontal Line Through Center of Cavity
 $Re = 100$ on a 30×30 Grid



(b)

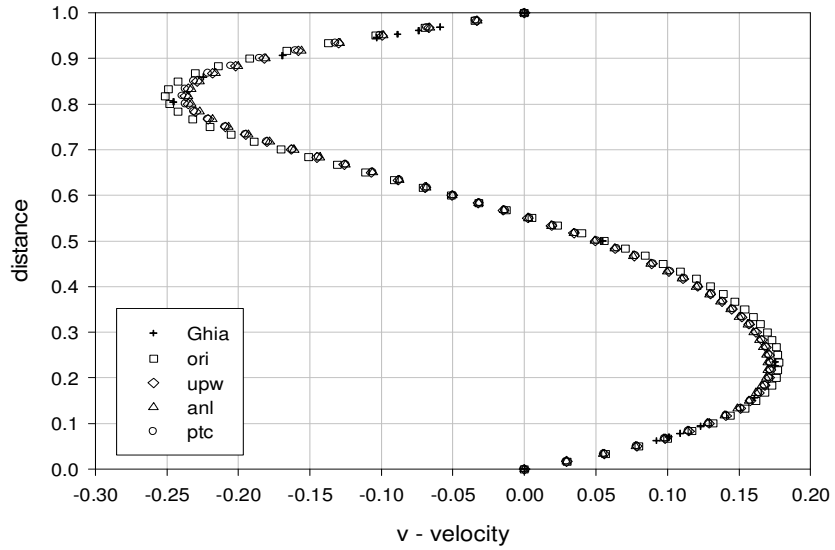
Figure 6.5: Velocities in the lid-driven cavity problem. Results of CHAD ori compare well with the benchmark results while all the upwind schemes produce slightly less accurate but similar results.

v -velocity along Horizontal Line Through Center of Cavity
 $Re = 100$ on a 30×30 Grid



(a)

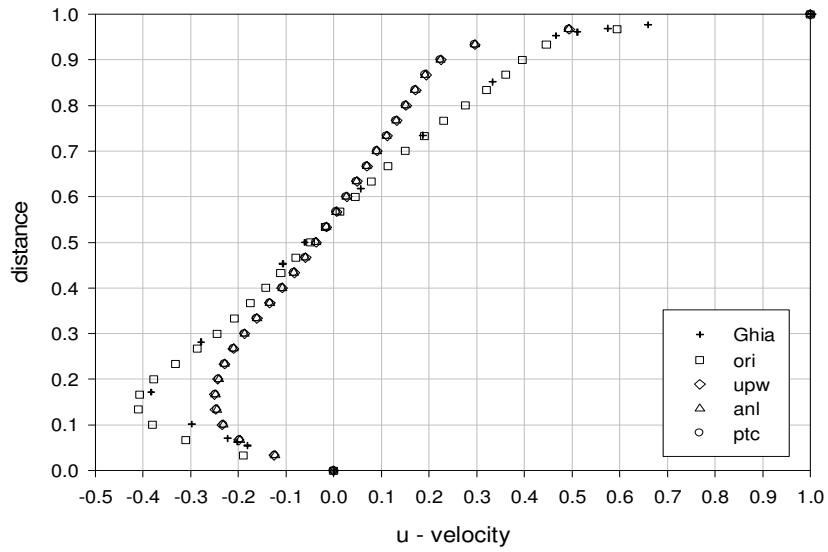
v -velocity along Horizontal Line Through Center of Cavity
 $Re = 100$ on a 60×60 Grid



(b)

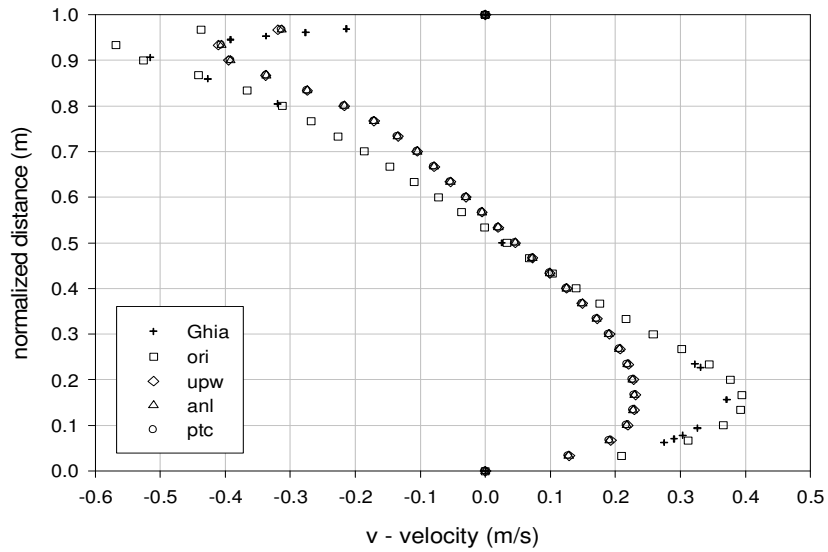
Figure 6.6: Velocities in the lid-driven cavity problem. CHAD ori performs quite well while all the upwind based schemes produce slightly less accurate results.

u -velocity along Vertical Line Through Center of Cavity
Re = 1000 on a 30 x 30 Grid



(a)

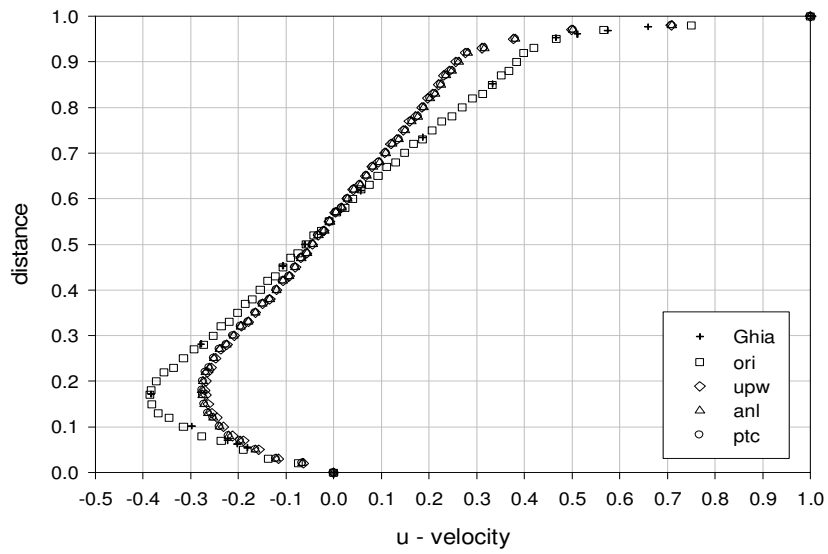
v -velocity along Horizontal Line Through Center of Cavity
Re = 1000 on a 30 x 30 Grid



(b)

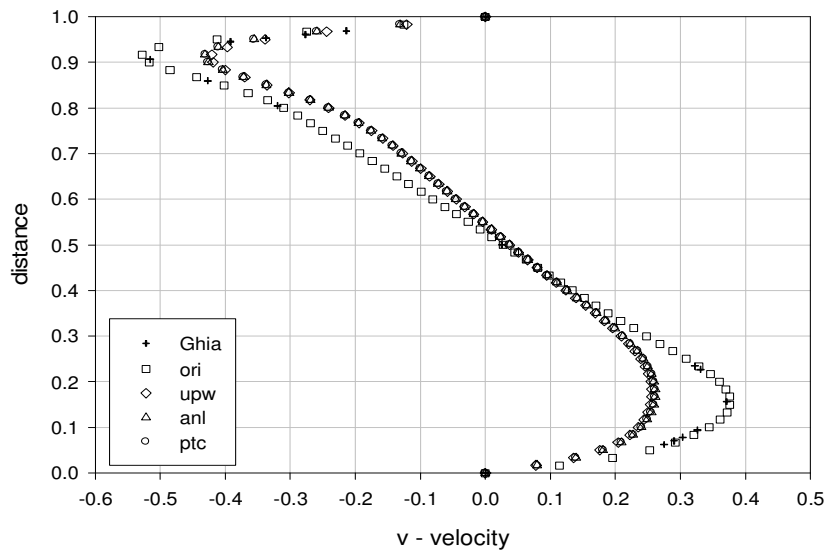
Figure 6.7: Velocities in the lid-driven cavity problem. That the $Re = 1000$ case is a more complicated problem is apparent at the peaks of the v -velocity profiles.

u -velocity along Vertical Line Through Center of Cavity
Re = 1000 on a 60 x 60 Grid



(a)

v -velocity along Horizontal Line Through Center of Cavity
Re = 1000 on a 60 x 60 Grid



(b)

Figure 6.8: Velocities in the lid-driven cavity problem. Doubling the grid density improves the FOU schemes, but still leads to a solution with excessive numerical diffusion for Re = 1000 case.

Since the results for the $Re = 1000$ case, obtained using the FOU numerical schemes even with the 60×60 mesh do not match well with the benchmark results, it was necessary to further refine the mesh to ensure that the numerical schemes do indeed lead to the correct solution as the mesh is refined. Hence, the problem was solved on a 120×120 grid. To visualize the convergence of these schemes with increasing mesh density, the results of 30×30 , 60×60 , and a 120×120 cell *ptc* simulations are plotted in Figure 6.9. Here it is apparent that the results obtained using the schemes based upon the FOU indeed continue to tend toward the benchmark solution with increasing grid density.

6.4.3 Lid-Driven Cavity Results Obtained Using PETSc and CHAD's Original Continuity Equation

As described in chapters three and four, CHAD's original continuity equation may be used to formulate a matrix system that can be solved using PETSc, while in chapter five we described the introduction of PETSc and a flag in the input file to activate the solution of CHAD's original continuity equation using PETSc.

The simulations described in the preceding section were repeated, with CHAD's original continuity equation this time solved with the PETSc library using GMRES, and are referred to as *ptc-o*. Hence, the only difference between *ptc* and *ptc-o* is the solver—built in GCR for the former and GMRES (through PETSc) for the latter. Results for u and v velocities on 60×60 and 120×120 grids are plotted in Figures 6.10 and 6.11.

Convergence of $Re = 1000$ PETSc Scheme on
30 x 30, 60 x 60, and 120 x 120 Grids

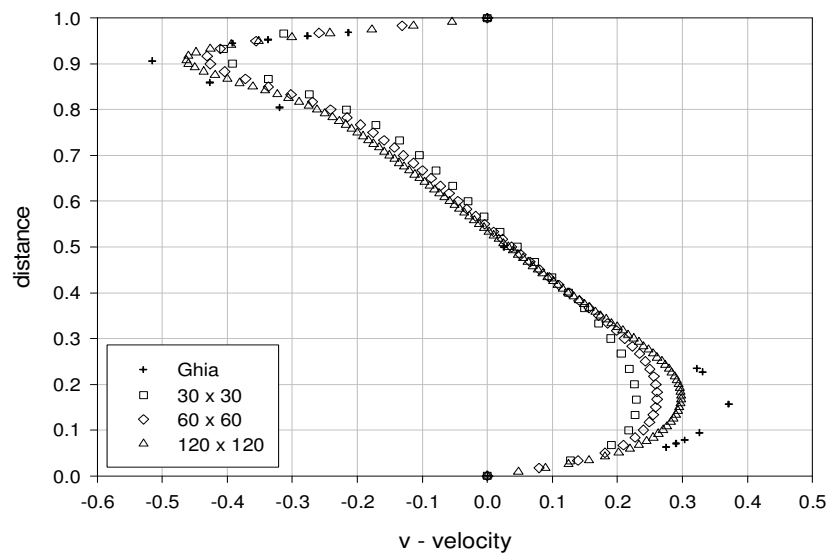
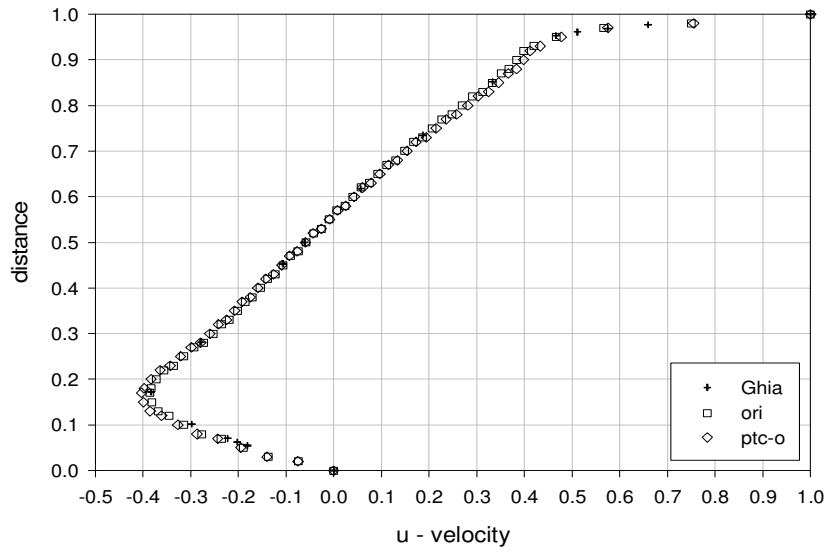


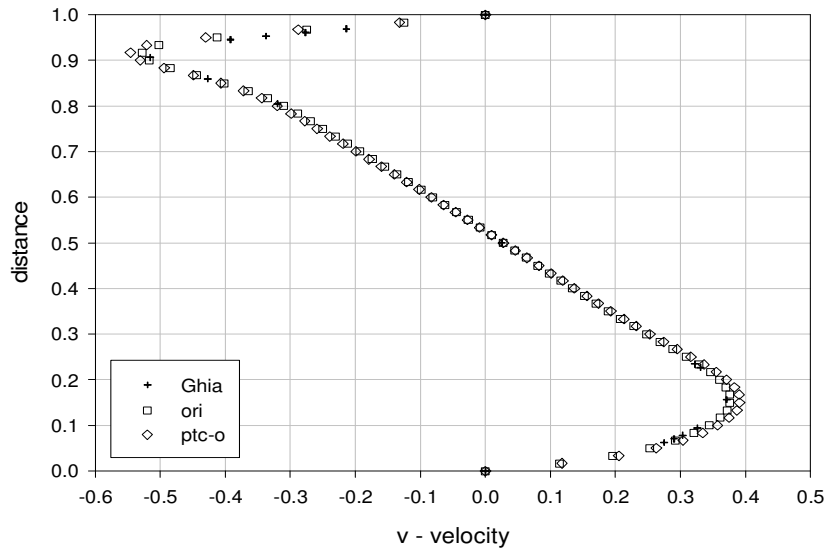
Figure 6.9: *The grid refinement shows continuous improvement in the accuracy of the solution.*

u -velocity along Vertical Line Through Center of Cavity
Re = 1000 on a 60x60 Grid



(a)

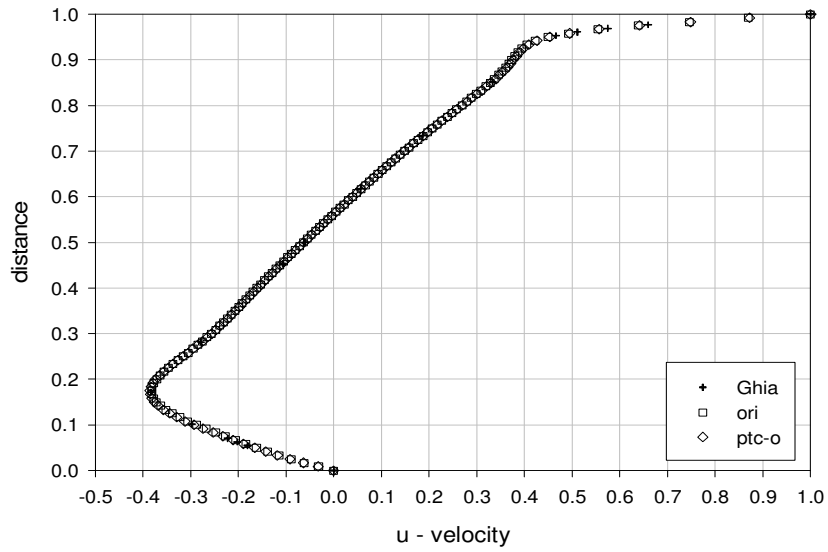
v -velocity along Horizontal Line Through Center of Cavity
Re = 1000 on a 60x60 Grid



(b)

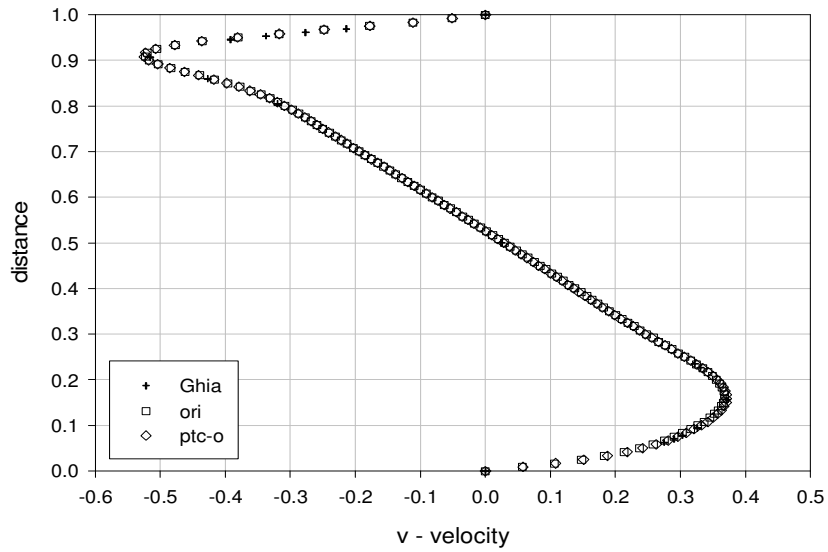
Figure 6.10: The solutions obtained using PETSc (GMRES) and GCR, based upon CHAD's original continuity equation.

u -velocity along Vertical Line Through Center of Cavity
 $Re = 1000$ on a 120×120 Grid



(a)

v -velocity along Horizontal Line Through Center of Cavity
 $Re = 1000$ on a 120×120 Grid



(b)

Figure 6.11: The solutions obtained using PETSc (GMRES) and GCR, based upon CHAD's original continuity equation.

These results show that the PETSc formulation (based upon the original continuity equation) leads to solutions that are nearly identical with the solutions obtained using the default GCR solver in CHAD. Additionally, these solutions are much more accurate than those obtained using the first order scheme, as is expected. Slight variations in the solutions may be due to insignificant terms that are neglected in the creation of the matrices, or from differences in the convergence criteria used by the PETSc solver (GMRES) and GCR.

6.5 Computational Controls in PETSc

Since a major goal of this dissertation is to link CHAD with the PETSc library to make available the wide range of solvers in PETSc, it is important to investigate the impact of various options available in PETSc when the library is used in conjunction with CHAD. These are investigated in this section.

Various options exist for user control of PETSc at runtime. One such control determines the convergence tolerances used by the PETSc solvers. Other options include choice of solvers or preconditioners, and even more detailed control of certain solvers, such as a restart parameter used to control the amount of memory used by the GMRES algorithm (which affects its performance). In this section, several controls are explained and their influence on CHAD solutions investigated.

|

6.5.1 Relative Tolerances

As discussed in chapters three and four, increasing convergence of the continuity equation is indicated by a drop in the nonlinear residual. In an iterative process in CFD codes changes in pressure are computed that would lead to a reduction in the linearized changes to the continuity equation. For example, when CHAD's internal GCR solver is used, it explicitly computes the linear residual after every iteration, and this residual is used to check for convergence. PETSc, on the other hand, does not require an explicit computation of the residual at every iteration, and in fact does not provide the capability to check the residual directly with a user application. Instead, different convergence tolerance controls are available in PETSc.

One way to control PETSc's convergence is to set a limit on the relative change in the residual between two successive iterations. If the relative change between two iterations is less than some predefined value, PETSc terminates the solution process and returns a solution (to CHAD). The default value for the relative tolerance convergence control is 10^{-5} . As it is the default value assigned to the PETSc solver, it is a value chosen for a wide variety of applications for which PETSc might be used. However, as the solution procedure in CHAD is an iterative process, it requires only an approximate solution to the continuity equation. Therefore it is quite likely that the default value for the relative tolerance convergence control in PETSc is too stringent for use in CHAD.

To determine whether the default value for relative tolerance (`rtol`) in PETSc was optimal for CHAD, the 30 x 30 and 60 x 60 cell lid-driven cavity problems for the

Re = 1000 case described in the previous section were solved multiple times using the original and reformulated continuity equations while varying the value of `rtol` using a command line option at runtime (`-ksp_rtol <value>`). Additionally, a fine mesh 120 x 120 cell problem was solved using the original continuity equation. Figure 6.12a shows the normalized times required to compute the hydrodynamic portion of the computations (momentum, energy, continuity, and stress deviator) for various tolerances using the reformulated continuity equation. A `rtol` of between 0.1 and 0.01 resulted in optimal times for both simulations indicating that for at least this particular problem it is an optimal value. Values higher than 0.1 indicate a lax solution of the continuity equation which ultimately requires more work in the iterative process. Values lower than 0.01 show that excessive effort was wasted in obtaining an accurate solution of the continuity equation *during the iterative process*, with the PETSc default of 10^{-5} resulting in exceptionally high computation times especially for the 60 x 60 cell problem.

Figure 6.12b shows the normalized hydrodynamic solution times using PETSc with the original continuity equation. Again, the fastest solution times were obtained when the relative tolerance was between 0.1 and 0.01. The solution was unstable for the 60 x 60 and 120 x 120 grids when the tolerance was larger than 0.1. This instability is not surprising as it is easy to imagine how an intermediate solution of the continuity equation that fails to sufficiently enforce mass conservation could lead to a divergence of a simulation.

The solutions obtained using various relative tolerances should be nearly identical. To check that various relative tolerances lead to nearly identical solutions, the lid driven cavity problem was solved for $Re = 1000$ case on a 60×60 grid using the reformulated continuity equation and relative tolerances of 0.9, 0.1 and 10^{-5} . Results are plotted in figure 6.13. It is clear that there is very little difference in the solutions.

Large differences in relative tolerances produce very minor differences in overall solution, while at the same time producing significant differences in computation time. Therefore, a CHAD user should consider beginning a computation by using relative tolerances around 0.1 and experiment with tighter tolerances if any convergence difficulties are encountered.

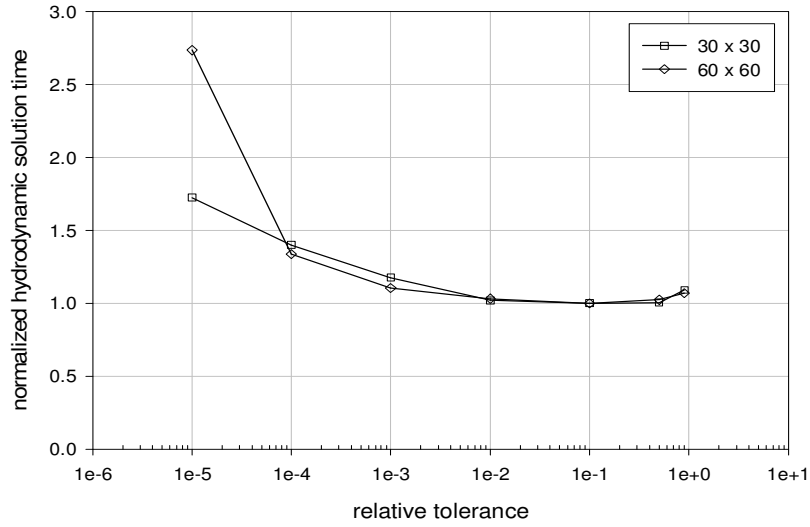
6.5.2 The GMRES Restart Control

In the Generalized Minimal Residual method (GMRES) a sequence of orthogonal vectors are generated. At each iteration, the vector that minimizes the residual satisfies the orthogonality condition

$$\underline{r}_k \perp span\left\{\underline{A}\underline{b}, \left(\underline{A}^2\right)\underline{b}, \dots, \left(\underline{A}^k\right)\underline{b}\right\}. \quad (6.3)$$

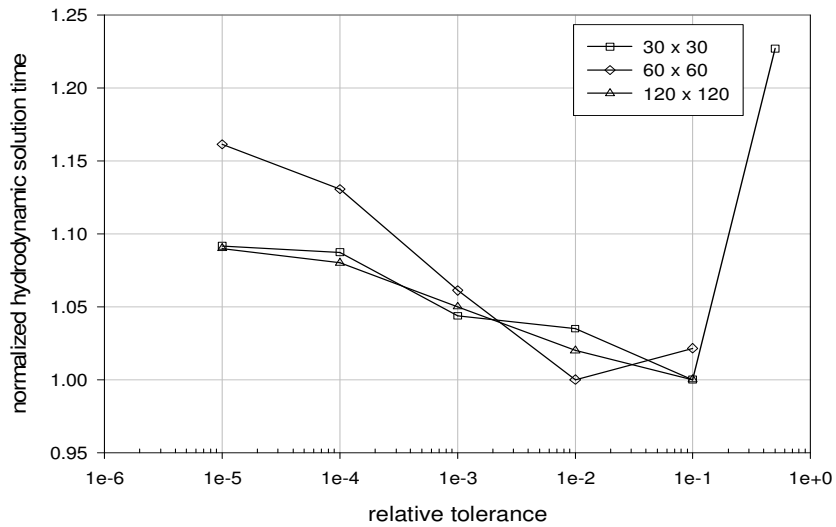
In non-symmetric cases, all of the orthogonal vectors must be retained as part of the iterative process.

Normalized Hydrodynamic Solution Times
using the Reformulated Continuity Equation and PETSc on Various
Grids for Different Relative Tolerances



(a)

Normalized Hydrodynamic Solution Times
using the Native Continuity Equation and PETSc on Various
Grids for Different Relative Tolerances



(b)

Figure 6.12: Normalized hydrodynamic solution times are optimized for fixed time steps when the relative tolerance of PETSc's GMRES solver is between 0.01 and 0.1. With some meshes, larger tolerances caused instabilities when using the original continuity equation.

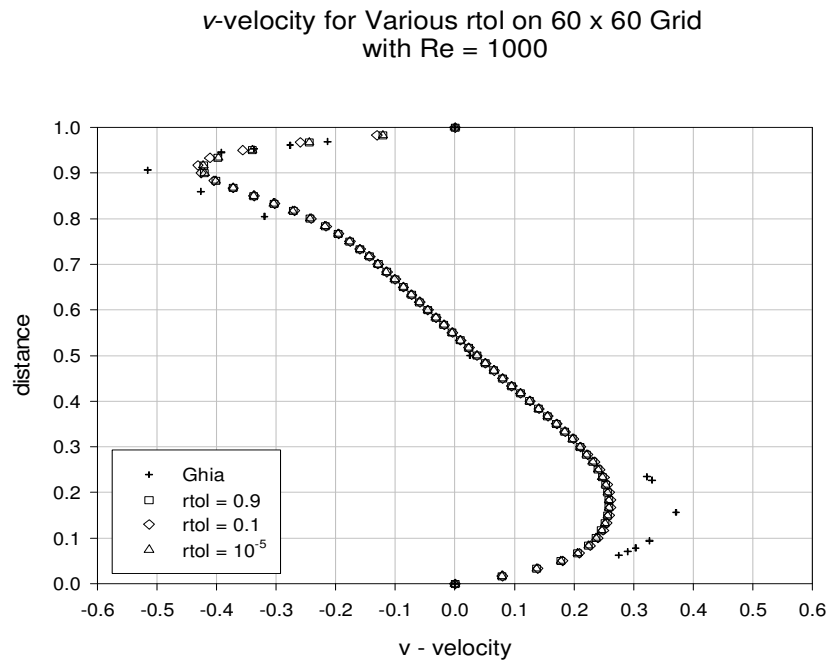


Figure 6.13: *Solutions obtained using various relative tolerances and the reformulated continuity equation are nearly identical.*

The storage of all previous iterate vectors therefore eventually becomes computationally expensive. To overcome this large memory requirement, “restarted” methods are used for GMRES (Greenbaum, 1997) in which the algorithm is simply restarted after a set number of iterations using the latest iteration vector as a new initial guess.

In PETSc, the ability to restart GMRES after a set number of iterations is available at run time via the command line using the flag `-ksp_gmres_restart` and specifying the number of iterations after which GMRES should restart the iterative process. Using a small numbers of saved iteration data, GMRES is expected to produce relatively poor solutions, especially with very large matrices, while using a large number of saved iteration data eventually becomes computationally costly.

To test the effects of the restart parameter on CHAD CPU time, the lid-driven cavity problem was solved on a 60 x 60 grid for the $Re = 1000$ case and using a relative tolerance of 0.1. The problems were all solved using four processors. Various values of the restart parameter between 3 and 45 were chosen. The PETSc default restart parameter is 30. The results of these simulations are shown in Figure 6.14 where normalized CPU time of the hydrodynamic part of the solution are plotted against the restart parameter. Overall, the effect of using the restart parameter is very small for the problem being tested. While some differences exist, they are quite small and the results offer no specific trend, though the highest computation time results when the restart

parameter is three—not an unexpected result. Larger values of the restart parameter can be expected to have little effect on the problems being solved here.

6.6 Solution Times Using CHAD and PETSc

One of the goals of this dissertation was to improve the performance of CHAD. This was to be accomplished through the implementation of PETSc to make high performance solvers available with CHAD which would thereby reduce overall computation times. At the onset of this work, it was believed that a reformulated continuity equation would be required to implement PETSc. Over the course of this work, it was found that the formulation of CHAD’s original continuity equation was sufficiently adequate to permit the development of a PETSc formulation. Because of this, the further testing of PETSc implementation described below is performed using only the original continuity equation. In these tests, several different lid-driven cavity simulations are carried out to examine the computation times required by CHAD, as well as to systematically carry out a performance study of different parts of the code.

Before we begin a detailed assessment of the performance of CHAD, a brief introduction to how CHAD reports its computation times, and what those times means, is provided.

Normalized Hydrodynamic Solution Times
for the $Re = 1000$ Lid Driven Cavity Problem on a 60×60
with Various Restart Parameters

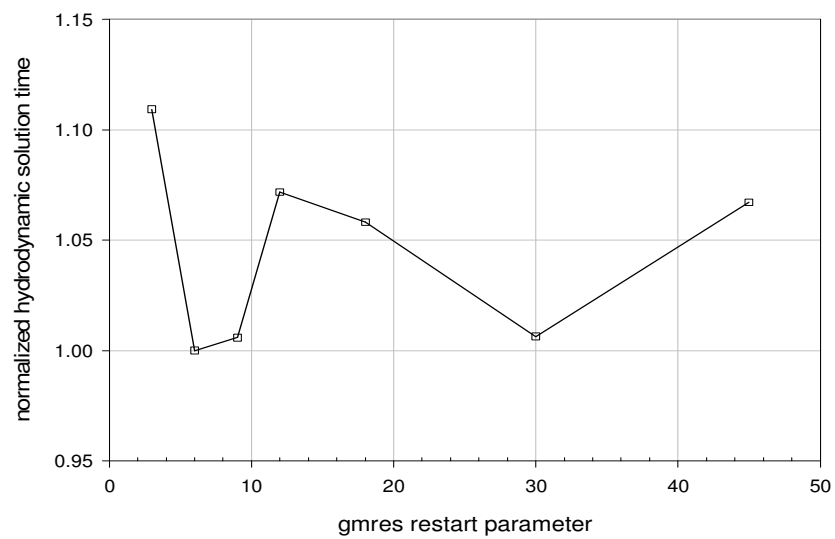


Figure 6.14: The effect of the restart parameter upon the normalized CPU times.

The most basic timing feature of CHAD is the wall clock time that is used to indicate the start and stop times of the simulation. These are given at the beginning and end of a CHAD output file.

```
This run is being made on:
05/11/2005 at 16:00:42
and
Run finished on:
05/11/2005 at 21:03:14.
```

At the end of the output file, a breakdown of the time is provided.

```
      CPU timing information:
Total problem CPU time      = 1.81515217E+04 s
  Total partitioning CPU time = 6.78990000E-02 s
  Total communication CPU time = 1.55786246E+04 s
  Total input/output CPU time = 3.16542176E+02 s
  Total mesh update CPU time  = 1.02764100E+00 s
  Total part tracking CPU time = 0.00000000E+00 s
  Total EOS CPU time         = 1.69297400E+01 s
  Total hydro CPU time       = 1.77543703E+04 s
    Node coupler             = 0.00000000E+00 s
    Boundary conditions       = 2.43622360E+01 s
    Interface quantities      = 2.90000000E-03 s
    Stress-deviator solver    = 0.00000000E+00 s
    Median-mesh quantities    = 2.56537174E+03 s
    Velocity solver           = 3.41236449E+03 s
    Temperature solver        = 8.99797156E+02 s
```

The total problem CPU (central processing unit) time generally corresponds to the start and finish times of the problem. It directly relates to time used by a processor to solve a given problem. While various processors on the *Reserv* computing cluster all have slightly independent processor speeds, CPU times remain within a few percent of each other between various processors. The partitioning time is the time required to

divide the domain for parallel computations, and is generally quite small compared to the overall time required to solve a problem. The communication time is the time required to compute *gather* and *scatter* operations. In multi-processor computations, this time includes both the time for communication between processors and moving it between associated data structures, while in single processor computations it merely describes the time required to transfer data between data structures. Because *gather* and *scatter* computations exist in most portions of the code, timings for individual sections of the code—such as the velocity solver—will also include the time required for communication. The mesh update section of the code is the time required for the calculation of mesh related quantities and is most significant when mesh motion exists. For the problems solved here, the time spent in the mesh motion stage is quite small. A similar situation exists for particle tracking. EOS is the time required to compute the equation of state, and is typically a small portion of the overall CPU time. The CPU time of the main hydrodynamic routines, which solve for velocity, temperature and mass conservation, are added up in the hydrodynamic CPU time. Because the solution to these equations is an iterative process representing the core of the code, they compose a large fraction of the overall CPU time. The individual components of the CPU time within the hydrodynamic routines, specifically the velocity, temperature, and continuity solver will account for a significant portion of the hydrodynamic CPU time. However, because computations are required for cell-face terms and other miscellaneous variables computed between iterations that are not part of the solver routines, the sum of the solver CPU times will be less than the total hydrodynamic solution time.

One last comment on solution times needs to be made before results are presented. When CHAD is run using its semi-implicit hydrodynamic mode (the mode in which most of its sample problems are solved) the PETSc implementations can only run with fixed time step, where as computations performed using CHAD's GCR solver are able to automatically vary the time step as a solution progresses. Therefore, when time steps are not restricted, a GCR based computation can compute with substantially lower overall solution times. This is an optimization issue where the time step controls are influenced by the convergence criteria associated with the solutions. For example, one of the controls which permit growth in the time step depends upon the number of inner iterations. As is described later in this chapter, the number of inner iterations of the PETSc implementation is strongly dependent upon relative tolerance used with the PETSc solver.

It is recognized that an optimization for PETSc of the time step growth controls that are currently optimized for CHAD's GCR solver could yield substantial improvements in overall solution times. While not explicitly computing the ideal optimization parameters of CHAD, certain performance enhancing features of CHAD are investigated independently. Combining these parameters together results in a multiple variable optimization problem of at least three variables and more realistically double that number, and in fact may be dependent upon the problem being solved itself. Investigating such optimizations would involve a substantial number of time consuming computations on a rather busy *Reserv* cluster at ANL. Heavy use of the ANL cluster by numerous groups did not allow such an optimization study. Therefore, *individual*

performance enhancing features of PETSc are investigated independently of each other. Additionally, comparisons between CHAD’s optimized GCR solver and several of PETSc’s optimized parameters coupled with its un-optimized time step controls are compared using fixed time steps for all computations.

Later in this chapter, we detail exactly why the PETSc computed solutions do not permit growth in the time step size in the semi-implicit mode, and present some results obtained using the implicit hydrodynamic mode.

6.6.1 Parallel and Serial Timings of CHAD using Fixed Time Steps

A “serial” version of CHAD, which does not attempt to utilize MPI in its calculations, is compared against a “parallel single processor” version of CHAD which was compiled to utilize MPI. In other words, a version of CHAD which contains overhead associated with MPI is compared against a version of CHAD that contains no overhead. Most simulations carried out using CHAD are expected to be run in ‘parallel’ mode on a single processor. This is due not only to simplifications in compiling the code (which must be done for every problem, and for any change in boundary conditions or source terms), but also because many simulations are initially performed using a coarse mesh. Therefore, it is informative to determine the amount of time ‘wasted’ using parallel simulations using a single processor. While this information may not necessarily offer a very clear image of how parallel computation time is spent within CHAD, it does provide a basis for comparisons later in this chapter. Table 6.4 shows the computation time required in seconds for various significant stages of a CHAD computation. The lid-

driven cavity problem described earlier in this chapter was used for the majority of the computations on 30 x 30, 60 x 60, and 120 x 120 grids.

Table 6.4: CPU Times for Single Processor Serial and “Parallel” Computations (seconds)

	30 x 30 Cavity		60 x 60 Cavity		120 x 120 Cavity	
	serial	‘parallel’	serial	‘parallel’	serial	‘parallel’
Total CPU Time	250.2	343.6	2107.3	2644.7	526836	598253
Communication time	75.4	161.3	590.1	1126.7	399224	409175
Velocity solver	30.9	41.5	306.4	364.9	71656	81126
Temperature solver	10.1	13.2	89.8	110.2	22170	24532
Continuity solver	87.2	122.7	537.2	726.7	159895	181153

In the solution of the cavity problem on a 30 x 30 grid, the overhead associated with using the parallel compilation results in an overhead of ~37%. Likewise, in the larger 60 x 60 grid, overhead amounts to ~25% of the total computation time. More significantly, the percentage of the total computation time devoted to parallel communication is only 30% and 28% for the serial mode computations, but 47% and 43% in the parallel computations. Though the overhead associated with MPI drops as a function of the total CPU time, the fraction of time spent in communication stays approximately the same. This is an indication that the communication time required by CHAD will comprise a substantial portion of the total CPU time. Additionally, the time required to compute the continuity equation increases by approximately 30-35% in both cases indicating a relationship between the amount of time spent in the continuity solver and the amount of communication associated with the continuity solver. In these cases,

the increase in communication time indicates that approximately 30% of the solution time of the continuity solver is dedicated to communication related operations. Similar deductions can be made about the temperature and velocity solvers, though we note that the overall increase in CPU times is significantly lower and therefore when optimizing a solver, the continuity solver is the first choice for optimization. This was of course a goal of this dissertation. The following section compares solution times obtained using PETSc to those obtained using CHAD's GCR solver.

6.6.2 Parallel PETSc Timings of CHAD using Two Continuity Equations and Fixed Time Steps

The reformulated and original continuity equations of CHAD were used to solve the lid-driven cavity problem described earlier on grids of 30 x 30, 60 x 60, and 120 x 120 grids. Solutions to these problems were obtained using PETSc's GMRES solver with the default restart parameter of 30, and a relative tolerance of 0.1. Time steps were fixed at 0.01 seconds for the 30 x 30 cell problem, 0.005 seconds for the 60 x 60 cell problem, and 0.0001 seconds for the 120 x 120 cell problem. All problems were run to a time of ten seconds. The results of these computations are shown in Table 6.5.

Clearly, the minimum total CPU time is obtained when PETSc is coupled with the original continuity equation, regardless of the number of processors being used. Additionally, the 30 x 30 cell simulations display the effects of parallel overhead at four processors, while the 60 x 60 cell simulations show the effects of overhead beginning at 8

Table 6.5: Continuity Solver, Communication and Total CPU Times for CHAD's Various Continuity Solver Arrangements (seconds)

		GCR			PETSc & orig continuity equation			PETSc & reformulated continuity equation		
model	# proc	cont eqn	total comm	total cpu	cont eqn	total comm.	total cpu	cont eqn	total comm	total cpu
30 x 30	1	57.5	161.3	343.5	19.8	104.4	232.6	26.76	111.1	292.6
	2	168.4	250.7	399.4	23.1	165.1	255.61	52.21	213.6	313.3
	4	171.5	195.3	345.1	22.83	138.4	216.0	41.26	143.5	225.78
	8	185.0	259.7	436.9	31.6	200.5	300.6	50.36	152.9	248.74
60 x 60	1	710.1	1132.6	2641.1	172.7	837.1	2121.1	187.7	861.0	1877.4
	2	626.0	1236.2	2018.8	146.1	879.4	1515.8	278.9	1178.8	1834.0
	4	434.3	861.5	1321.5	101.4	603.4	945.5	208.2	828.3	1213.9
	8	369.9	930.5	1351.8	88.9	629.7	881.9	187.6	879.1	1197.7
120 x 120	1	181153	409175	598253	87400	386350	501565	--	--	--
	2	118686	304025	407634	28079	245050	315612	--	--	--
	4	89209	221491	277897	17009	136055	180667	31742	74648	86435
	8	45467	194804	225041	11155	134733	157901	5477.7	15578	18151

processors. From the data presented in Table 6.5, we can conclude that an optimal loading per processor should be (as a rough guideline) no less than 1,000 nodes and 2,300 connections per processor.

Solutions were not obtained using the reformulated continuity equations on the 120 x 120 cell grid for one and two processors due to the excessively long computation times.

6.6.3 Parallel Solvers and Preconditioners in CHAD

One of the most touted aspects of the PETSc toolkit is the large array of solvers and preconditioners that are readily available at runtime through command line options. The default solver in both CHAD and PETSc is GMRES. However, PETSc also includes

such solvers as the conjugate gradient (CG) (Hestenes, 1952), conjugate residual (CR, also known as Orthomin), transpose-free quasi-minimum residual solver (TFQMR) and the stabilized bi-conjugate gradient method (BiCGSTAB). The latter two schemes are variations of the BiConjugate Gradient method (Trefethen, 1997).

For preconditioners, PETSc uses the incomplete lower-upper (ILU) preconditioner (Meijerink, 1977) for uniprocessor simulations and block Jacobi preconditioning with each block using ILU for multiprocessor simulations. Table 6.6 shows the results for the same 60 x 60 grid lid driven cavity problem solved on four processors using a variety of solvers and preconditioners. The intent here was not to perform a detailed study of which solvers are best for any particular problem (a decision that should be investigated by the user for their own particular model and geometry), but more to illustrate the availability of the different packages provided by CHAD and offer preliminary evidence of how these perform within CHAD.

Table 6.6: Solver and preconditioner performance for a 60 x 60 cavity problem.

<i>one processor</i>					
Solver	Preconditioner	time (s)	Solver	Preconditioner	time (s)
GMRES	None	2102	CR	None	2085
GMRES	ILU	2082	CR	ILU	2088
GMRES	Jacobi	2079	CR	Jacobi	2109
GMRES	LU	2378	CR	LU	2374
GMRES	SOR	2056	CR	SOR	unstable

<i>Table 6.6 continued:</i>					
<i>four processors</i>					
Solver	Preconditioner	time (s)	Solver	Preconditioner	time (s)
GMRES	Block Jacobi	945	GMRES	Additive Schwarz	1172
CR	Block Jacobi	942	CR	Additive Schwarz	1105
CG	Block Jacobi	1052	CG	Additive Schwarz	unstable
BiCGSTAB	Block Jacobi	unstable	BiCGSTAB	Additive Schwarz	934
TFQMR	Block Jacobi	1013	TFQMR	Additive Schwarz	1097
LSQR	Block Jacobi	unstable	LSQR	Additive Schwarz	unstable

6.6.4 CHAD Performance Using PETSc and Variable Time Steps

Thus far comparisons between CHAD and PETSc have been limited to simulations conducted using fixed sized time steps. In practice, time step sizes in CHAD are typically allowed to float between maximum and minimum values prescribed by the user in the input file. In most CHAD sample problems, however, time step sizes are fixed rigidly and thus our selection of time step sizes for sample problems has been conducted using fixed time steps that are stable for both PETSc simulations based upon both continuity equations, as well as standard CHAD simulations with its original continuity solver.

When solving new problems we do not have the luxury of knowing optimal time step sizes, and hence other restrictions, such as the Courant number of the flow field, must be considered in determining the time step. However, a better way of dealing with this issue is to begin a simulation with a small time step and allow it to increase in size up

until some limit upon the time step is reached. This method works fine with the default solver and time step controls provided with CHAD. However, when attempts are made to solve these same problems using PETSc and CHAD's semi-implicit algorithm, difficulties arise.

The computation of the time step is performed in routine `timestep.FF`. For the semi-implicit hydrodynamic mode, the time step size is a function of an optimal number of outer iterations *and* an optimal number of inner iterations during the *first* outer iteration. This works fine for the GCR solver as it checks for convergence at every iteration against CHAD's built-in convergence criteria given by equations (3.34, 3.35 and 3.37). However, as described in section 6.5.1 above, the PETSc solver is unable to check for convergence in the same way that CHAD does and therefore is susceptible to performing an excessive number of iterations, thereby unnecessarily increasing the overall computation time. Worse yet, with PETSc performing an excessive number of computations, the time step algorithm incorrectly determines that the time step is too large and attempts to compensate by reducing the next time step size—a behavior that is quite possibly exactly opposite to what should occur were the time steps computed based upon the residuals to the problem!

By specifying a rather lax relative tolerance, the number of continuity iterations required for 'convergence' is easily reduced. However, too few iterations of the continuity equation may cause insufficient convergence and an increase in outer iterations.

CHAD's standard implicit solver does not depend rigidly on the number of continuity iterations. For this reason, it may be possible to use the PETSc solver while retaining the use of CHAD's automatic time step advancement controls. To test this theory, the $120 \times 120 \times 1$ cavity problem was solved again, this time using implicit hydrodynamics. Because of the limited availability of the *Reserv* cluster at ANL (cluster has been exceptionally busy), the simulations were run in serial mode on a 1.6 GHz, 256 Mb RAM linux machine at UIUC. For this reason, simulation times are not directly comparable to those available elsewhere in this chapter.

The initial time step size for the problem was 0.001 seconds. The minimum allowable time step was set at 0.0001 seconds. The maximum allowable time step was set at 0.0125 seconds. Both simulations were run until a problem time of ten seconds. A relative tolerance of 0.1 was used with the PETSc solver.

The simulation run using the GCR solver completed in $1.62\text{E}+5$ seconds, while the solution based on the PETSc solver was completed in $5.15\text{E}+4$ seconds. As before, communication times comprised a substantial portion of the total CPU time, accounting for 78% of the total problem time for the GCR solver and 58% of the time required when using the PETSc solver.

Since solution obtained using the PETSc solver are nearly identical to those obtained using GCR, and the solution times obtained using PETSc are substantially

lower, it substantiates the assertion made earlier that the iteration count control on the continuity equations first outer iteration prevents the semi-implicit solver from allowing time steps to grow. If the iteration count control had inhibited the growth of the time step, the PETSc calculation would have been restricted to a very small time step size, and the calculation would have taken substantially longer to execute.

6.6.5 PETSc Performance Summary & Log Info

PETSc includes a utility to report summaries of certain actions performed over the course of a simulation. These results can be accessed by providing the command line option `-log_summary` at run time. This summary reports the number of specific PETSc events (or operations) performed over the course of a computation. Unfortunately, however, there is no hierarchy as to the PETSc operations being performed, so the summary is of limited use (Balay, 2000). Nevertheless, it is of some use in determining where PETSc spends a majority of the time or effort in a computation. For example, it could be used to determine if the majority of the time spent in PETSc occurs during the application of the preconditioner or the loading of the matrix.

Table 6.7 shows selected portions of the log summary for a 60 x 60 x 1 grid lid driven cavity computation on four processors. The summary includes the time it takes for particular operations within PETSc. For example, MatMult is the time spent doing matrix multiplications. VecCopy is the amount of time copying vectors, VecSet is the amount of time spent setting up those vectors etc. In this particular simulation, which was run to a time of 10.0 seconds using fixed time steps of 0.005 seconds, there are a

minimum of 2,000 outer iteration passes through the continuity solver. More if during any cycle more than one outer iteration computation is required. From the log summary it can be seen that there are 2071 *linear solves* performed, meaning there were 71 more *solves* performed than there were cycles and therefore 71 more outer iterations than there were cycles. That means there are at least 2071 parallel matrix creations and 4142 parallel vector creations at a minimum.

More detailed information is available by supplying the command line option `-log_info`, which provides verbose information about the work being performed by PETSc during each iteration. For example, the memory allocation requirement during matrix assembly is provided when this option is specified, which has a rather substantial impact on PETSc performance. While the results from this log information output are not discussed here, through the use of the log summary, efforts have been made to optimize PETSc with the CHAD simulations conducted in this work. For example, the maximum number of non-zero matrix entries per row (discussed in Chapter 5) has been optimized using this information.

Table 6.7: Selected PETSc Log Summary Results for the Driven Cavity Problem

Event	Count	sec max	Total			Mflops/s
			% time	% flops	% reductions	
VecMDot	17661	3.60E+00	0	25	27	511
VecNorm	19792	2.92E+00	0	4	29	101
VecScale	19792	6.94E-02	0	2	0	2122
VecCopy	60	5.60E-04	0	0	0	0
VecSet	23994	1.35E-01	0	0	0	0
VecAXPY	2191	1.50E-02	0	0	0	2174
VecMAXPY	19792	4.66E-01	0	20	0	4518
VecAssemblyBegin	2071	8.31E-01	0	0	9	0
VecAssemblyEnd	2071	4.87E-03	0	0	0	0
VecScatterBegin	17721	2.03E+00	0	0	0	0
VecScatterEnd	17721	3.69E+00	0	0	0	0
VecNormalize	19792	3.03E+00	0	6	29	146
MatMult	17721	6.46E+00	1	20	0	222
MatSolve	19792	1.13E+00	0	18	0	1148
MatLUFactorNum	2071	6.86E-01	0	2	0	195
MatILUFactorSym	2071	5.38E-01	0	0	3	0
MatAssemblyBegin	2071	3.07E+00	0	0	06	0
MatAssemblyEnd	2071	5.78E+00	1	0	21	0
MatGetOrdering	2071	1.19E-01	0	0	6	0
PCSetUp	4142	1.61E+00	0	2	9	83
PCSetUpOnBlocks	2071	1.42E+00	0	2	9	95
PCApply	19792	1.52E+00	0	18	0	853
KSPGMRESOrthog	17661	4.05E+00	0	51	26	908
SLESSetup	4142	1.79E+00	0	2	9	75
SLESSolve	2071	1.48E+01	2	98	55	484

Chapter 7: Incompressible and Buoyant Flows

7.1 Equations of State in CHAD

As described in previous chapters, CHAD was developed for compressible flows, and option does not exist in CHAD to model completely incompressible flows. Compressible equation of state in CHAD is inadequate to model incompressible flows. In pressurized water reactors, water is commonly modeled as an incompressible fluid. Therefore, to make CHAD more useful to the nuclear industry, an incompressible equation of state is needed in CHAD. CHAD currently contains a variety of equations of state: perfect gas, ideal gas, isothermal, SESAME (LANL), Miegru and Osborne equations of state. To facilitate the simulation of incompressible fluids, an incompressible equation was added to CHAD.

7.2 Adding an Incompressible Equation of State

The routine `read_input` was modified to read data specific to the incompressible equation of state. First, it was modified to read the keyword `incompress` from the input file, which specifies that the incompressible equation of state will be used. As no input specifications for CHAD exist [Sahota-2001] the reader should refer to a sample input file provided with the CHAD distribution for the proper location for the keyword. The `read_input` was also modified to read a second line of data specific to the incompressible equation of state. The specific heat capacity and the density of the fluid are specified on this new line in the data file. Though the density of

the fluid is often specified in the mesh file or the input file, the density specified in the material property section of the input file will overwrite those densities.

With the input routines modified to read the properties for the incompressible equation of state, the actual incompressible equation of state is then added to the module `eos_module_d` in the subroutine `eos`. Here, the derivative of density with respect to both pressure and temperature are set to zero (equations 7.1, 7.2), while density itself is set to the value read in the material properties section of the input file (again, the reader should refer to a sample input file for the proper location),

$$\left. \frac{\partial \rho}{\partial p} \right|_T = 0 \quad (7.1)$$

$$\left. \frac{\partial \rho}{\partial T} \right|_p = 0 \quad (7.2)$$

The change in specific enthalpy with respect to pressure at constant temperature is set as the value of the specific heat capacity of the fluid read in the material properties section of the input file (equation 7.3).

$$\left. \frac{\partial h}{\partial T} \right|_p = c_p \quad (7.3)$$

Specific enthalpy is computed as the product of the specific heat capacity and temperature as given in equation (7.4).

$$h = c_p T \quad (7.4)$$

Though density should not change within the code due to equations (7.1) and (7.2), it is rigorously set to the value given in the input file for all nodes.

$$\rho = \rho_{input} \quad (7.5)$$

Currently no advantage is taken of the simplifications that arise when the flow is treated as incompressible. For example, though the time derivative in the continuity equation is identically zero, it is still evaluated and “found” to be zero in the incompressible case. Additionally, computations involving changes in density are still performed, even though the results are known to be zero. Ideally, an incompressible version of CHAD should be developed by preprocessing directives at compilation time and sections of the code should be blocked off as is common practice in many commercial CFD codes. Significant computation time could be saved if the incompressible equation of state was optimized.

As was described in section 3.5.1, the GCR solver in CHAD is susceptible to numerical difficulties when nearly incompressible flows are modeled due to “division by zero” errors. This was because the approximation to the reciprocal of the derivative of the residual with respect to pressure contained terms that depended only upon changes in density, as shown in equation (7.6)

$$d_v = \left[\frac{\partial \rho}{\partial p} \Big|_{T,v}^n \delta p_v + \frac{\partial \rho}{\partial T} \Big|_{p,v}^n \delta T_v \right]^{-1} \quad (7.6)$$

Therefore, the use of an incompressible equation of state with the default GCR preconditioner is not possible without further changes to the code. [Though a module containing preconditioning routines is supplied with the CHAD distribution, there are no references within the code to these subroutines.] As the framework for computing the momentum terms in equation (3.6) is found in the PETSc matrix formulation and is also

found in the same subroutine, it is a simple matter to compute one extra term to provide a more accurate representation of the inverse of the diagonal, and is given as

$$d_v = \left[\frac{\partial \rho}{\partial p} \Big|_{T,v}^n dp_v + \frac{\partial \rho}{\partial T} \Big|_{p,v}^n \delta T_v + \frac{\Delta t}{V_v^{n+1}} \sum_{\alpha} \rho_{\alpha} \delta \tilde{\mathbf{u}}_{\alpha} \cdot \mathbf{A}_{\alpha}^{n+1} \right]^{-1} \quad (7.7)$$

where the diagonal term now incorporates changes in mass due to flow between cells. This formulation of the reciprocal of the derivate of the residual for the continuity equation with respect to pressure was added to CHAD for use when the incompressible equation of state is specified or when the reformulated continuity equation is used. PETSc internally computes its own preconditioner and therefore does not require the preconditioners shown above.

7.2.1 Testing the Incompressible Equation of State

The incompressible equation of state was tested using the same lid-driven cavity problem described in chapter six. As the default lid-driven cavity problem supplied with the CHAD distribution is nearly incompressible—showing a compressibility of less than $1/10^{\text{th}}$ of a percent—results for this problem are expected to closely agree with the results for the incompressible case.

Figures 7.1a and 7.1b show the results (u and v velocities) of the incompressible lid-driven cavity problem on grids of $30 \times 30 \times 1$ and $60 \times 60 \times 1$ cells using the PETSc solver with the incompressible equation of state. They are compared against solutions obtained by Ghia et al. and those obtained using the default GCR solver and a perfect equation of state. In both figures it is evident that the solutions obtained using the

incompressible equation of state are nearly identical to those obtained using the perfect equation of state.

Computation times for simulations using the incompressible equation of state are similar to those obtained using the compressible EOS. This is expected as this work only restricts the density from changing. It does not prevent computations which involve changes in density from being performed. In other words, changes in density are still being computed even though the results of these computations result in no change of density.

7.3 Buoyant Flows

The incompressible equation of state described previously in this chapter was expanded upon to account for buoyancy. For most problems that CHAD was developed to model, forces due to buoyancy are small in comparison to other forces. However, there exist a number of natural convection problems that are of interest to nuclear engineers, such as the natural convection in spent fuel pools. To enable CHAD to model such cases, a buoyancy model was added to CHAD.

v -velocity along Horizontal Line Through Center of Cavity
 $Re = 1000$ on a 30×30 Grid
 with an Incompressible EOS

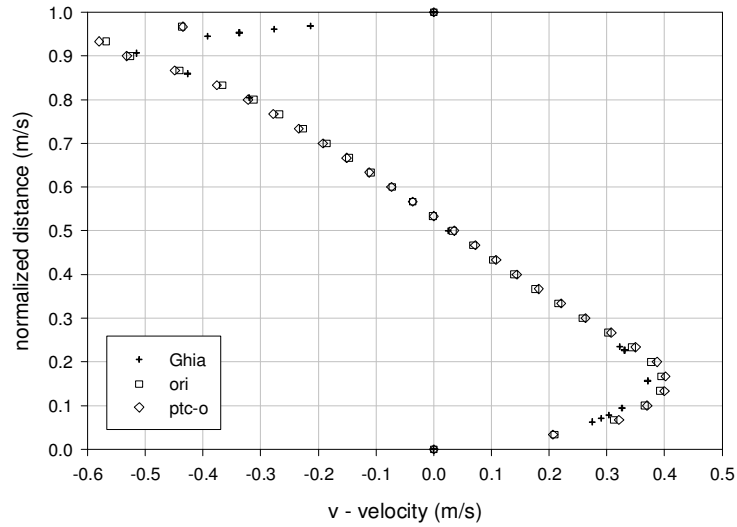


Figure 7.1a: Solutions to the lid-driven cavity problem using an incompressible equation of state and PETSc (ptc-o) are similar to those obtained using the perfect equation of state and the GCR solver (ori).

v -velocity along Horizontal Line Through Center of Cavity
 $Re = 1000$ on a 60×60 Grid
 with an Incompressible EOS

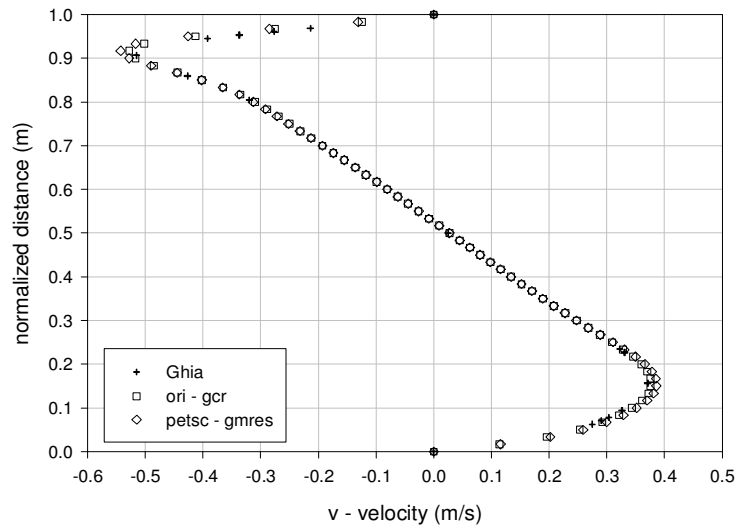


Figure 7.1b: On a more refined grid, the incompressible equation of state and PETSc also produce results similar to the GCR solver and the perfect equation of state.

One common buoyancy model is the Boussinesq approximation, named after Valentin Joseph Boussinesq (1842-1929). In this approximation, variations in density are ignored except in the gravity term. In the discretized momentum equation, buoyancy effect is included by multiplying the force due to gravity by the change in nodal temperature about a reference temperature times a volumetric expansion coefficient

$$F_b = \rho \mathbf{g} \beta (T_v - T_{ref}) \quad (7.8)$$

The new discretized momentum equation (2.14) then becomes

$$\begin{aligned} & \frac{\rho_v^{n+1} \mathbf{u}_v^{n+1} V_v^{n+1} - \rho_v^n \mathbf{u}_v^n V_v^n}{\Delta t} + \sum_{\alpha} \rho_{\alpha} \mathbf{u}_{\alpha} [\tilde{\mathbf{u}}_{\alpha} \cdot \mathbf{A}_{\alpha}^{n+1}] + \rho \mathbf{g} \beta (T_v - T_{ref}) \\ & + \sigma_T \sum_{\alpha} \left[\left(\frac{2}{3} \rho K \right)_{\alpha}^n - \left(\frac{2}{3} \rho K \right)_v^n \right] \mathbf{A}_{\alpha}^{n+1} + \frac{1}{a^2} \sum_{\alpha} [p_{\alpha} - p_v^{n+1}] \mathbf{A}_{\alpha}^{n+1} \quad (7.9) \\ & = \sum_{\alpha} (\tau_{\alpha} - f_v \tau_v) \cdot \mathbf{A}_{\alpha}^{n+1} + \mathbf{N}_v + (\mathbf{S}_u)_v \end{aligned}$$

The volumetric expansion coefficient is a new keyword required by the buoyancy model and is specified in the input file as `vexpansion`. Additionally, the gravity term must also be specified in the input file as its individual vector components `gravx`, `gravy`, and `gravz`.

7.3.1 Testing the Boussinesq Approximation Part 1

The natural convection of a fluid in a cavity is a standard problem for comparing the numerical solution of buoyancy driven flows and is discussed by de Vahl Davis and Jones in 1983 (de Vahl Davis and Jones) and again by de Vahl Davis (de Vahl Davis) in further detail. The problem consists of a square cavity of dimension D with insulated top and base and vertical walls of specified temperatures T_1 and T_2 . This model is depicted in Figure 7.2. The fluid inside the cavity has a Prandtl Number of 0.71.

This problem was solved using CHAD for Rayleigh Numbers of 10^3 , 10^4 , 10^5 , and 10^6 where the Rayleigh Number is given by

$$Ra = \frac{\beta g \Delta T D^3}{\kappa \nu} \quad (7.10)$$

and κ is the thermal diffusivity and ν is the kinematic velocity.

The solutions obtained using CHAD on 30 x 30 grid are first compared qualitatively against the benchmark solutions. Then, a quantitative comparison between CHAD and the benchmark solutions are performed. These comparisons are conducted using a variety of grid sizes.

The contours for u velocity for different Rayleigh Numbers agree well with the bench mark solutions and are found in Figures 7.3 a-d. Of note are contours in the upper left and bottom right corners which are believed to be artifacts of CHAD's treatment of corner nodes, a subject which is discussed in more detail later in this thesis.

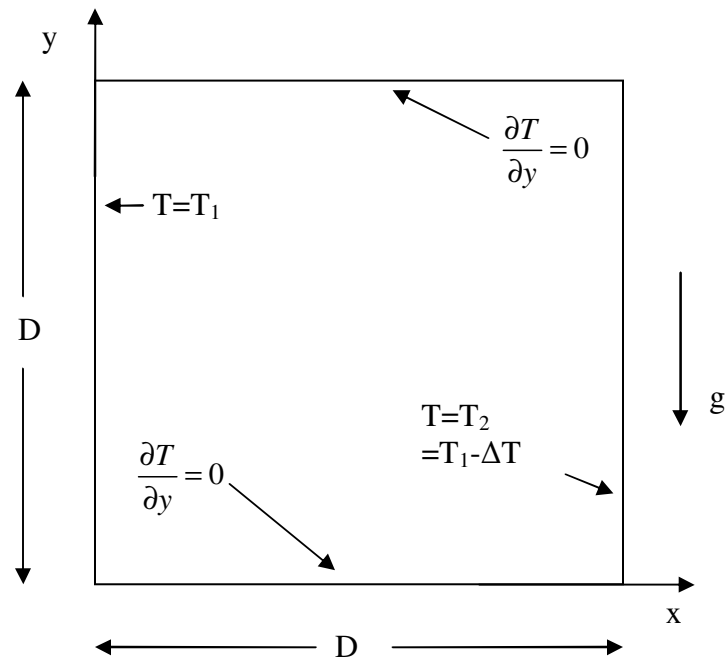


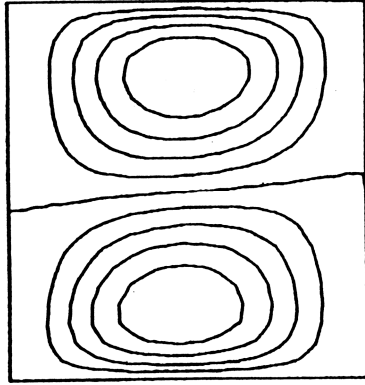
Figure 7.2: *Illustration of the 2-D cavity used to model natural convection. The left and right walls of the cavity are set to prescribed temperatures, while the top and the bottom surfaces are insulated.*

Additionally, the size of the grid used in the CHAD computations is evident along the fixed-temperature walls by a straight line near the wall surface. This is the same contour as found at the upper left and bottom right corner nodes. For the most difficult case of $Ra = 10^6$ more sizable variations in the solutions are evident. A finer grid of 90 x 90 cells was used to compute this case more accurately. The results of that case are found in Figure 7.3d-2. There, as expected, the computed solution more closely matches the bench mark solution.

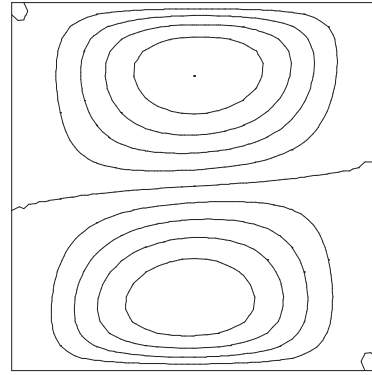
The contours for v velocity, found in Figures 7.4 a-d, display a larger variation than those of the u component of velocity. In particular, the contours along the top and bottom (insulated) walls follow the node nearest to the wall which is used to calculate its temperature using a first order approximation.

Differences in the temperature contours found in Figures 7.5a-d are much more difficult to visualize as there are few defining characteristics to the profiles. Rayleigh numbers of 10^5 and greater, display gradients in the horizontal direction in the temperature profiles though they are generally captured by the CHAD simulations. As seen with the velocity contours, the high Rayleigh number temperature simulations require finer grids to capture the profiles.

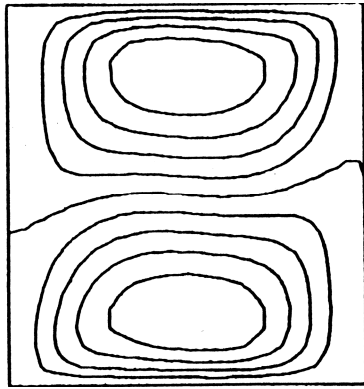
The final set of qualitative comparison was performed using vorticity and is found in Figures 7.6a-d. Like the comparisons with u and v velocities, and temperature, the vorticity profiles generally agree well with the bench mark solution.



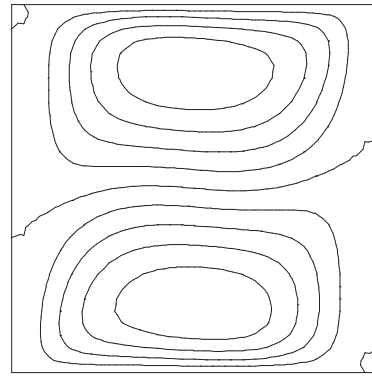
a) Benchmark u velocity: $Ra = 10^3$



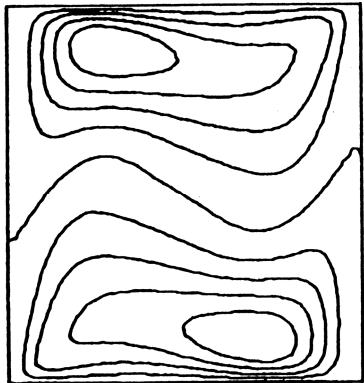
a) CHAD u velocity: $Ra = 10^3$



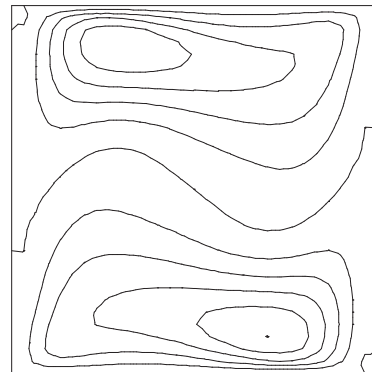
b) Benchmark u velocity: $Ra = 10^4$



b) CHAD u velocity: $Ra = 10^4$

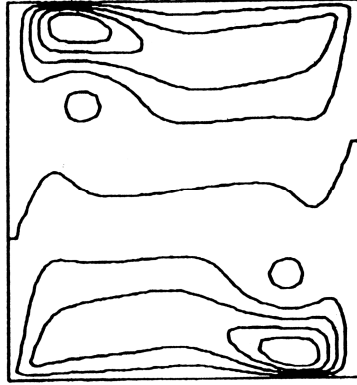


c) Benchmark u velocity: $Ra = 10^5$

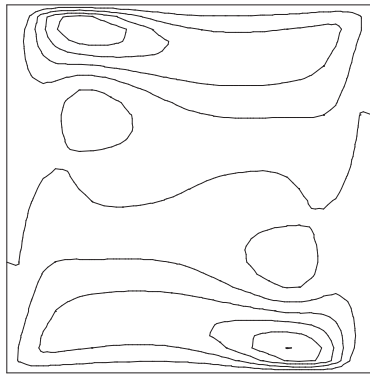


c) CHAD u velocity: $Ra = 10^5$

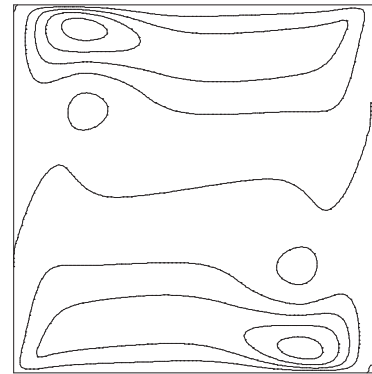
Figures 7.3: U -velocity contours for benchmark and CHAD solutions at various Rayleigh numbers



d) Benchmark u velocity: $Ra = 10^6$

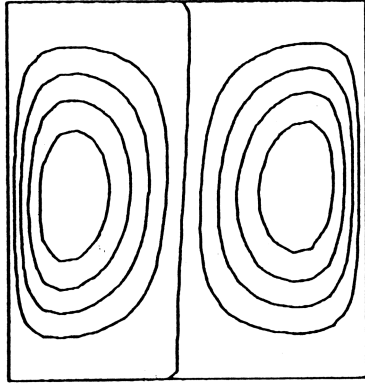


d-1) CHAD u velocity: $Ra = 10^6$

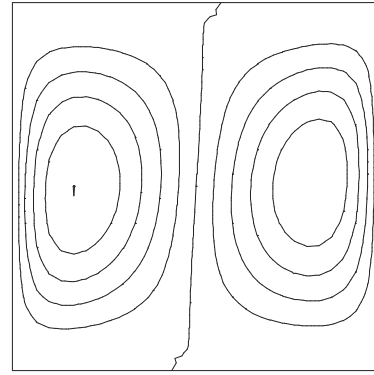


d-2) CHAD u velocity: $Ra = 10^6$ on a refined grid

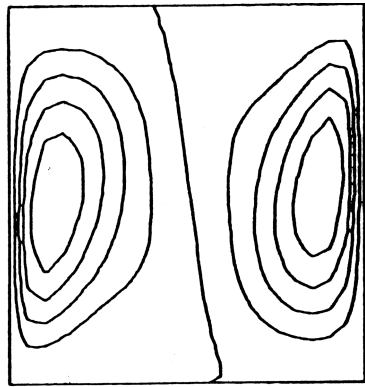
Figures 7.3: U -velocity contours for benchmark and CHAD solutions at various Rayleigh numbers (cont'd)



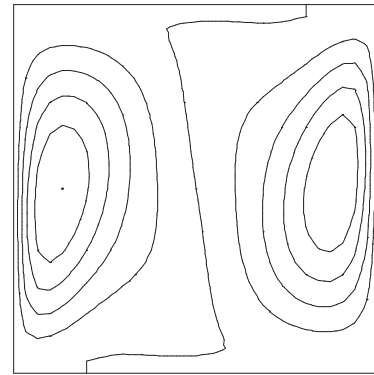
a) Benchmark v velocity: $Ra = 10^3$



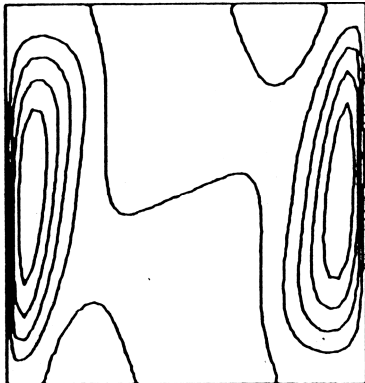
a) CHAD v velocity: $Ra = 10^3$



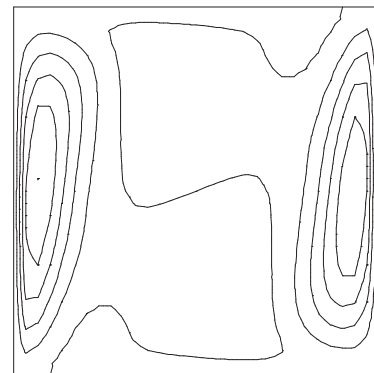
b) Benchmark v velocity: $Ra = 10^4$



b) CHAD v velocity: $Ra = 10^4$

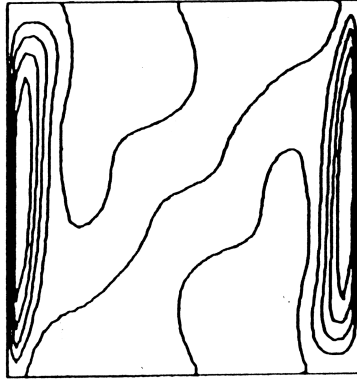


c) Benchmark v velocity: $Ra = 10^5$

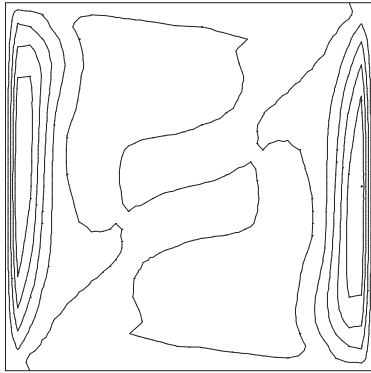


c) CHAD v velocity: $Ra = 10^5$

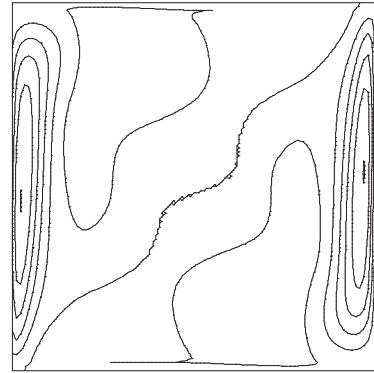
Figures 7.4: V -velocity contours for benchmark and CHAD solutions at various Rayleigh Numbers



d) Benchmark v velocity: $Ra = 10^6$



d-1) CHAD v velocity: $Ra = 10^6$



d-2) CHAD v velocity: $Ra = 10^6$ on a refined grid

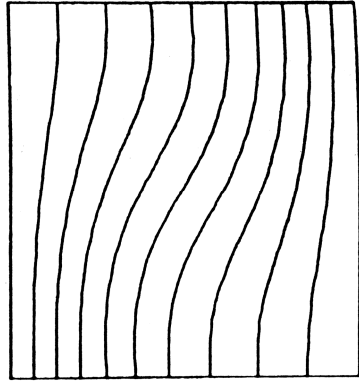
Figures 7.4 (continued): V -velocity contours for benchmark and CHAD solutions at various Rayleigh numbers

Only at high Rayleigh Numbers do variations in profiles become readily apparent. In particular, at $Ra = 10^5$ and $Ra = 10^6$ bubble shaped contours develop near the centers of the cavity.

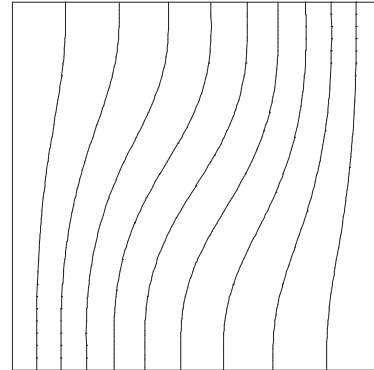
Quantitative comparisons were made against benchmark solutions for average, maximum, and minimum Nusselt numbers along the hot wall. Additionally, along the vertical mid-plane of the cavity, maximum u velocity and average Nusselt number were compared, as was the maximum vertical velocity along the horizontal mid-plane. These quantities are tabulated in Table 7.1. The bench mark solutions for these terms are given in non-dimensional form, which are defined in Table 7.2.

Table 7.1: Non-dimensional quantities used in the natural convection problem

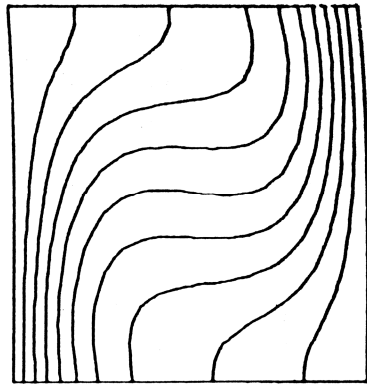
Description	
u_{\max}	The maximum horizontal velocity on the vertical mid-plane of the cavity, together with its location
v_{\max}	The maximum vertical velocity on the horizontal mid-plane of the cavity, together with its location
$Nu_{1/2}$	The average Nusselt number of the vertical mid-plane of the cavity
Nu_0	The average Nusselt number on the vertical boundary at $x = 0$
Nu_{\max}	The maximum value of the local Nusselt number on the vertical boundary at $x = 0$, together with its location
Nu_{\min}	The minimum value of the local Nusselt number on the vertical boundary at $x = 0$, together with its location



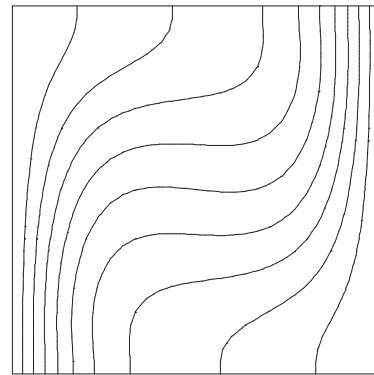
a) Benchmark temperature: $Ra = 10^3$



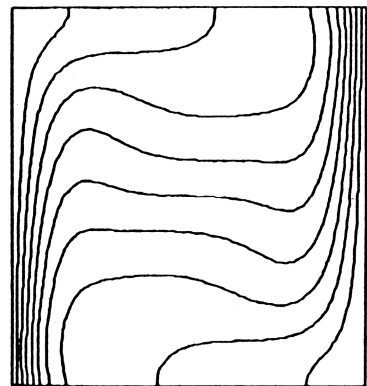
a) CHAD temperature: $Ra = 10^3$



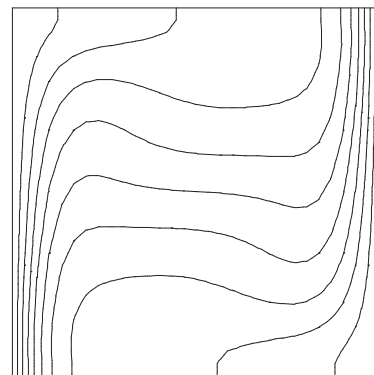
b) Benchmark temperature: $Ra = 10^4$



b) CHAD temperature: $Ra = 10^4$

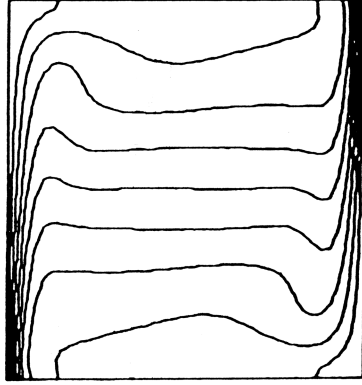


c) Benchmark temperature: $Ra = 10^5$

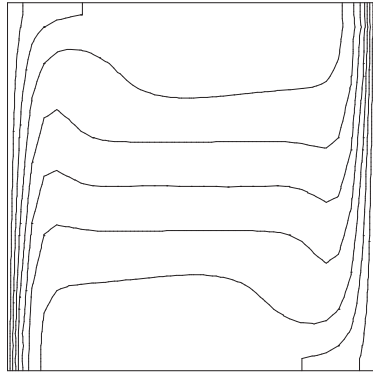


c) CHAD temperature: $Ra = 10^5$

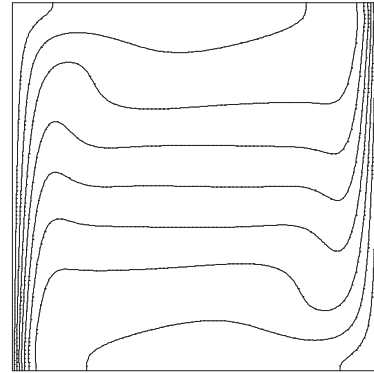
Figures 7.5: Temperature contours for benchmark and CHAD solutions at various Rayleigh numbers



d) Benchmark temperature: $Ra = 10^6$

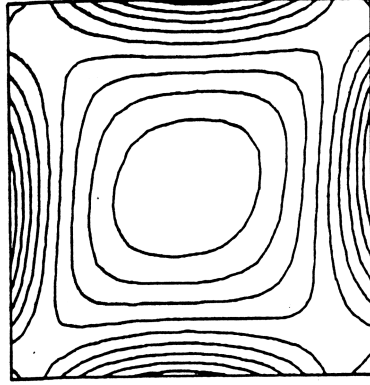


d-1) CHAD temperature: $Ra = 10^6$

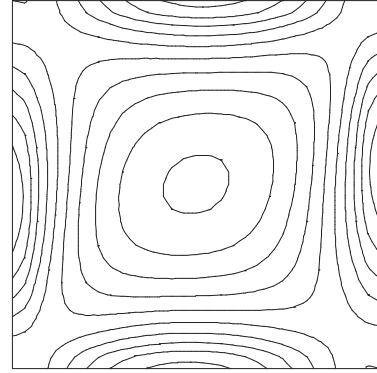


d-2) CHAD temperature: $Ra = 10^6$ on a refined grid

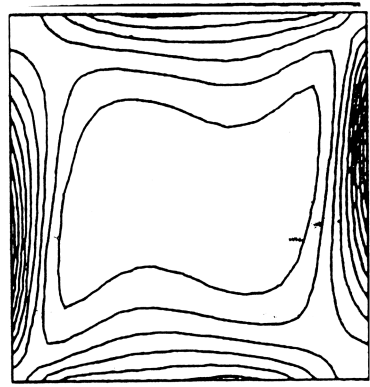
Figures 7.5 (continued): Temperature contours for benchmark and CHAD solutions at various Rayleigh numbers (cont'd)



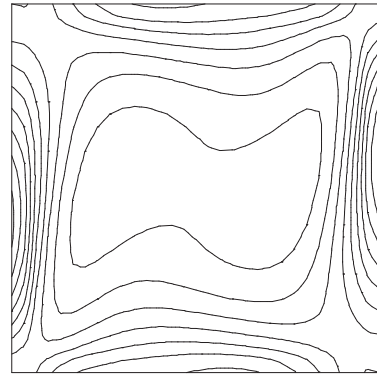
a) Benchmark vorticity: $Ra = 10^3$



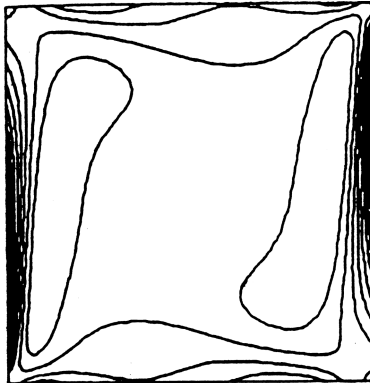
a) CHAD vorticity: $Ra = 10^3$



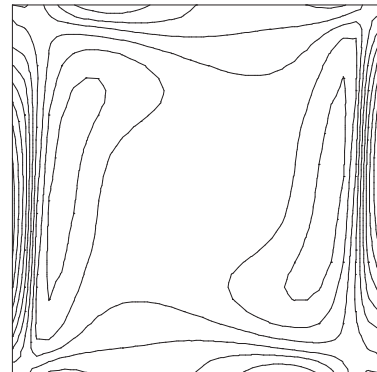
b) Benchmark vorticity: $Ra = 10^4$



b) CHAD vorticity: $Ra = 10^4$

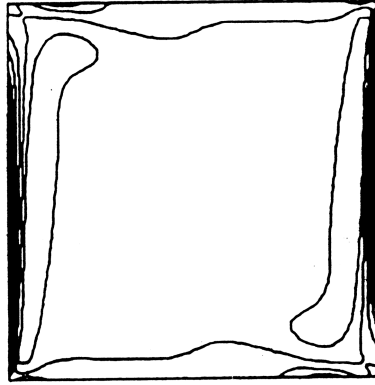


c) Benchmark vorticity: $Ra = 10^5$

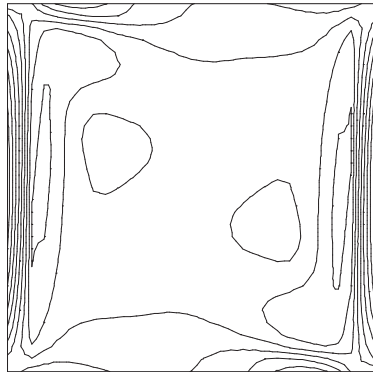


c) CHAD vorticity: $Ra = 10^5$

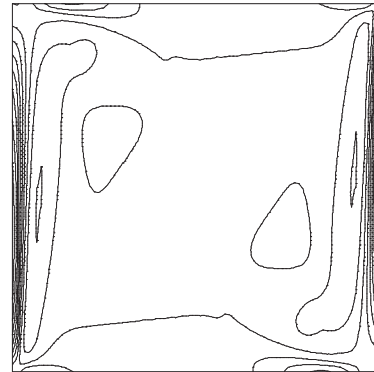
Figures 7.6: Vorticity contours for benchmark and CHAD solutions at various Rayleigh numbers



d) Benchmark vorticity: $Ra = 10^6$



d-1) CHAD vorticity: $Ra = 10^6$



d-2) Benchmark vorticity: $Ra = 10^6$ on a refined grid

Figures 7.6 (continued): Vorticity contours for benchmark and CHAD solutions at various Rayleigh numbers (cont'd)

Table 7.2: Non-dimensional quantities used in the natural convection problem

Non-dimensional notation	
Non-dimensional temperature	$T = \frac{T - T_2}{T_1 - T_2}$
Non-dimensional coordinates	$x, y = \frac{x}{D}, \frac{y}{D}$
Non-dimensional velocities	$u, v = \frac{uD}{\kappa}, \frac{vD}{\kappa}$

At any point in the cavity, the local heat flux in the horizontal direction is defined as

$$Q(x, y) = uT - \frac{\partial T}{\partial x} \quad (7.11)$$

using the non-dimensional variables. Based upon this definition, the average Nusselt number along any vertical mid-plane is defined as

$$\overline{Nu}_x = \int_0^1 Q(x, y) dy. \quad (7.12)$$

Because the velocity at a wall is zero, the average Nusselt number at the hot wall is given by

$$\overline{Nu}_0 = \int_0^1 \left. \frac{\partial T}{\partial x} \right|_{x=0} dy. \quad (7.13)$$

The definitions of equations (7.11-7.13) enable us to quantitatively compare Nusselt numbers and velocities within the cavity. These comparisons were made for Rayleigh numbers of 10^3 , 10^4 , 10^5 and 10^6 on 10×10 , 20×20 , 30×30 , 40×40 , 50×50 , 70×70 and 90×90 cell grids, with one cell in the z direction.

To compute the Nusselt number at the hot wall, a quadratic polynomial curve was used to fit the data points. For the computation of the Nusselt number at the cavity mid-plane, a linear interpolation was used to compute the change in temperature across a node. For this calculation, a quadratic fit was not found to be any more useful.

Table 7.3 shows the results of 30 x 30 cell computations of CHAD and the bench mark results obtained by de Vahl Davis. In general the results obtained using CHAD agree well with the bench mark results, with the lower Rayleigh number results displaying a better agreement.. The locations for maximum velocities and Nusselt numbers are at the nearest node and no effort was made to interpolate the results. We do note however, that the bench mark results are obtained using a form of interpolation. To demonstrate that our results obtained using CHAD converge towards the bench mark results, the data for $Ra = 10^5$ is displayed in Table 7.4 for the various grid densities listed above.

Table 7.3: Natural Convection on a 30 x 30 Grid using CHAD

	Rayleigh Number							
	10^3		10^4		10^5		10^6	
	Bench mark	CHAD	Bench mark	CHAD	Bench mark	CHAD	Bench mark	CHAD
u_{\max}	3.649	3.603	16.178	16.078	34.73	36.23	64.63	75.84
@ $y =$	0.813	0.800	0.823	0.833	0.855	0.867	0.850	0.867
v_{\max}	3.697	3.664	19.617	19.345	68.59	69.09	219.36	219.02
@ $x =$	0.178	0.167	0.119	0.133	0.066	0.066	0.0379	0.0333
$Nu_{1/2}$	1.118	1.140	2.243	1.959	4.519	4.371	8.799	7.958
Nu_0	1.117	1.126	2.238	2.268	4.509	4.677	8.817	9.496
Nu_{\max}	1.505	1.524	3.528	3.547	7.717	7.882	17.925	17.787
@ $y =$	0.092	0.050	0.143	0.067	0.081	0.067	0.0378	0.033
Nu_{\min}	0.692	0.696	0.586	0.621	0.729	1.042	0.989	2.958
@ $x =$	1	1	1	1	1	1	1	1

It can be noted that the simulations on the 10 x 10 and 20 x 20 grids lead to quite poor agreement with the benchmark results and good agreement for Nu_0 is achieved at 70 x 70 grid, while good agreement with benchmark results for $Nu_{1/2}$ is found at 90 x 90 grid. Similarly, the maximum and minimum velocities and their locations display convergence towards the bench mark results, which have an estimated error of 0.3% (de Vahl Davis), as the grid is refined. One possible explanation for the slow convergence of Nu_0 might be the quadratic fit used in the computation. Another is that the rest of the flow field requires a tighter convergence before the effects are realized at the hot wall. A third reason might be related to the boundary condition treatment at the hot wall.

The order of convergence is a measure of how fast the error decreases when the grid density increases. By taking the natural log of both the error and the grid density, the order of the error can be determined as the slope of a line when the log-log values are plotted. The convergence rate of CHAD was determined using $Nu_{1/2}$ solutions for $Ra = 10^5$ along the cavity mid plane. The slope of the line is 2.58, indicating better than 2nd order convergence.

Table 7.4: Convergence of key parameters for $Ra = 10^5$

Rayleigh Number = 10 ⁵				
	Nu ₀		Nu _{1/2}	
Grid Size	Bench mark	CHAD	Bench mark	CHAD
10 x 10	4.509	4.109	4.519	2.787
20 x 20		4.796		4.096
30 x 30		4.677		4.371
40 x 40		4.655		4.457
50 x 50		4.545		4.483
70 x 70		4.551		4.498
90 x 90		4.563		4.510
	Nu _{max} @ y =		Nu _{min} @ y =	
Grid Size	Bench mark	CHAD	Bench mark	CHAD
10 x 10	7.717 0.081	5.728 0.000	0.729 1.0	1.432 1.0
20 x 20		7.644 0.000		0.979 1.0
30 x 30		7.882 0.067		1.042 1.0
40 x 40		7.910 0.075		0.853 1.0
50 x 50		7.813 0.100		0.807 1.0
70 x 70		7.728 0.071		0.786 1.0
90 x 90		7.768 0.078		0.775 1.0
	U _{max} @ y =		V _{max} @ x =	
Grid Size	Bench mark	CHAD	Bench mark	CHAD
10 x 10	34.73 0.855	28.11 0.800	68.59 0.066	49.86 0.200
20 x 20		33.45 0.800		62.47 0.050
30 x 30		36.23 0.867		69.09 0.066
40 x 40		34.88 0.850		67.40 0.075
50 x 50		34.61 0.860		67.72 0.060
70 x 70		34.63 0.857		68.02 0.071
90 x 90		34.63 0.855		68.43 0.066

Convergence of the Average Nusselt Number Along
Cavity Vertical Mid-plane for $Ra = 10^5$

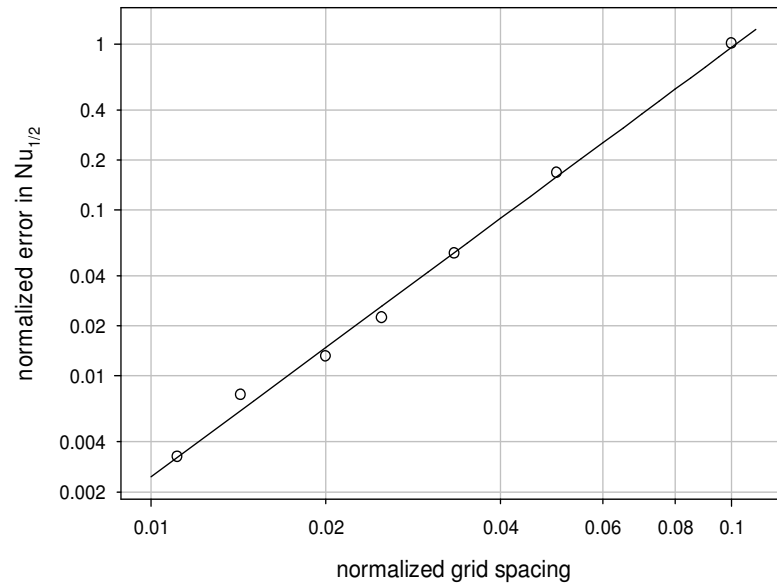


Figure 7.7: Convergence of the error in the average Nusselt number along the cavity vertical mid-plane. The slope of the line in the log-log plot is 2.58 indicating at least a second order scheme.

7.3.2 Testing the Boussinesq Approximation: Part 2

A second problem was developed to test the incompressible equation of state and Boussinesq model implemented in CHAD. The problem is similar in geometry to the natural convection problem described in the previous section. However, the results of this calculation are not compared against benchmark reference solutions. Instead, the commercial CFD code Fluent was used to obtain reference solutions. Fluent is widely used in a variety of fields and is a commonly accepted engineering tool for fluid flow applications. Like CHAD, Fluent is a finite volume based CFD code. But unlike CHAD, Fluent uses a vertex based cell nodalization, whereas CHAD uses a cell centered scheme.

The geometry of this natural convection problem is similar to that of the lid-driven cavity problems—a 0.3 m x 0.3 m two-dimensional cavity with density meshes ranging from 30 x 30 to 120 x 120 cells. The bottom third of the cavity is filled with a uniform source term of 10 W/m^3 . Because of CHAD's intermediate mesh, it utilizes a different control volume than Fluent. Therefore, to keep the total source term the same, explicit FORTRAN coding was included in CHAD to account for the difference. Cells that were bisected by edge of the source region in Fluent (between the source region and the non-heated region) were adjusted to use the appropriate fraction of the total source term. This causes a small portion of the source term to be applied in a half-cell width region in which it is not applied in Fluent. However, the total source term is the same. The source term adjustment was done to preserve an identical nodalization between the codes. The alternative would have been to use different grids for CHAD and Fluent. As the total source term is the same, but the area in which the source term is slightly

different, minor differences in the solution may be expected for coarse grids. As grids are refined, and the error between the source areas decreases, any error in the solutions due to the difference in areas should correspondingly be expected to decrease.

The cavity contains water of standard properties, and a gravity field of -9.81 m/s^2 was applied throughout. The top wall of the cavity was kept at a fixed temperature of 300 K, while all other walls were insulated. All walls were no-slip in nature.

The problem was solved using Fluent on a 30 x 30 cell grid. Under-relaxation factors of 0.95 were used for the energy equation, while 0.3 was used for the pressure term and 0.7 for the momentum term. Additional solutions were obtained using 60 x 60, 90 x 90, and 120 x 120 cell grids. For the denser meshes, different under relaxation factors were required in order to obtain convergence. Notably, the 120 x 120 cell problem was found to be particularly difficult to converge. For this problem, under relaxation factors of 0.7 for pressure and 0.3 for momentum were required.

The same problem was solved using CHAD and the results were post processed using GMV. It was found that GMV was incapable of representing variations in variables beyond just a few decimal places. This is a known bug (Ortega-b) To circumvent this limitation, the material properties were customized to result in both temperature and velocity fields that could be post processed. The Fluent simulations were repeated using a density of 1000 kg/m^3 , a specific heat of 0.04182 J/kg K , a dynamic viscosity of 0.3, 0.001 W/m K for thermal conductivity, and 5×10^{-4} as a

thermal expansion coefficient. At convergence, these properties result in a Reynolds number of approximately 25.

The velocity magnitude vectors from a Fluent 120 x 120 cell simulation are shown in Figure 7.8 to illustrate the resulting flow field for this calculation. The vectors have been reduced in magnitude for the sake of clarity. The peak velocity occurs in the center of the cavity in the upward direction with a magnitude of approximately 0.024 m/s. Two centers of recirculation exist nearly between the top and the bottom of the cavity, and between the cavity centerline and the side walls. Velocities are very low in the bottom corners of the cavity.

The temperature distributions in a 60 x 60 cell simulation carried out using Fluent are plotted at various horizontal cross sections to illustrate the temperature profiles and grid spacing in the cavity. These cross sections are at 3 cm increments beginning at the bottom of the cavity. These cross sections are shown in Figure 7.9. The maximum temperature rise across the cavity is approximately 20 degrees.

Figure 7.10 shows a comparison between CHAD and Fluent solutions on a 60 x 60 cell grid. CHAD predicts temperatures predominantly between 0.7 and 1.0 degrees K hotter than Fluent across any given line in the model with the exception of the bottom most

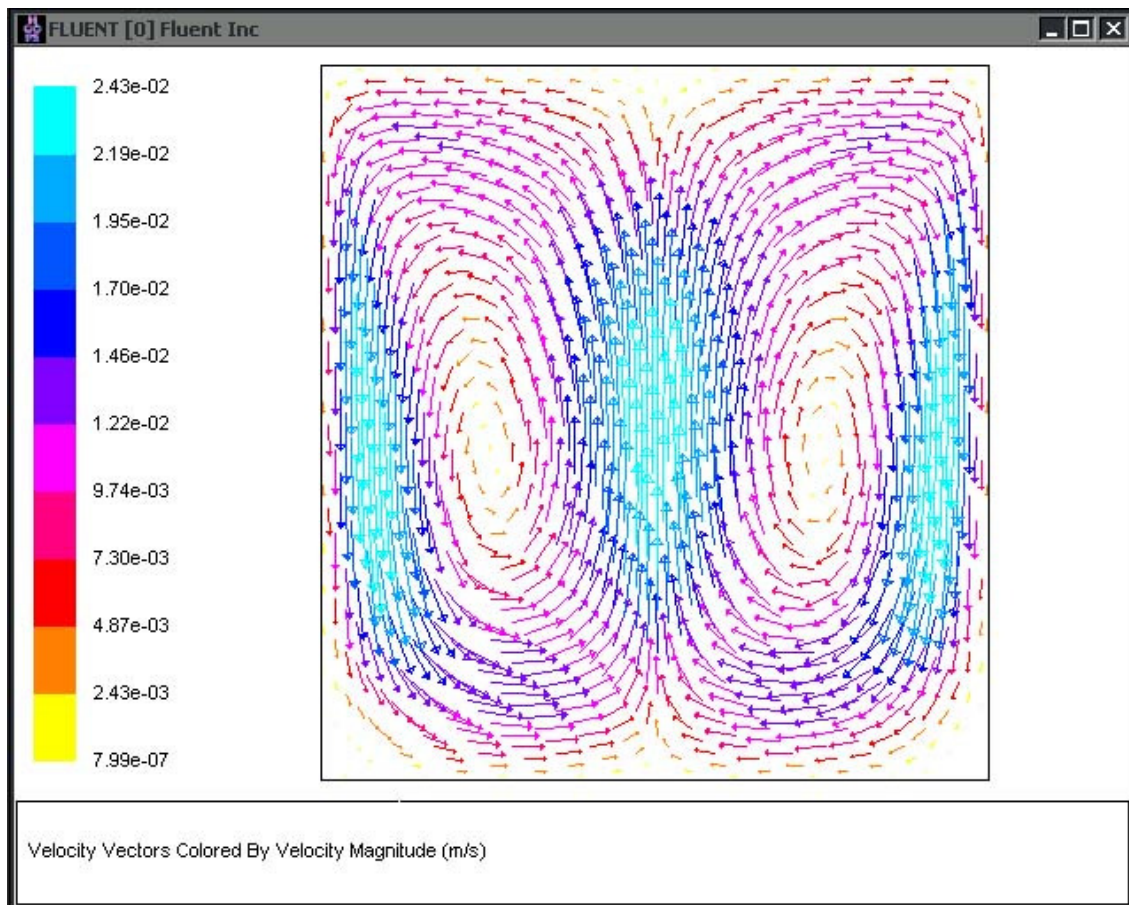


Figure 7.8: Velocity vectors in a square, volumetric energy source term driven cavity. A source term of 10 W/m^3 is in the bottom third of the cavity. Two centers of recirculation exist on either side of the vertical midplane of the cavity.

Fluent Calculated Temperatures Across Heated Cavity
for Various Axial Levels

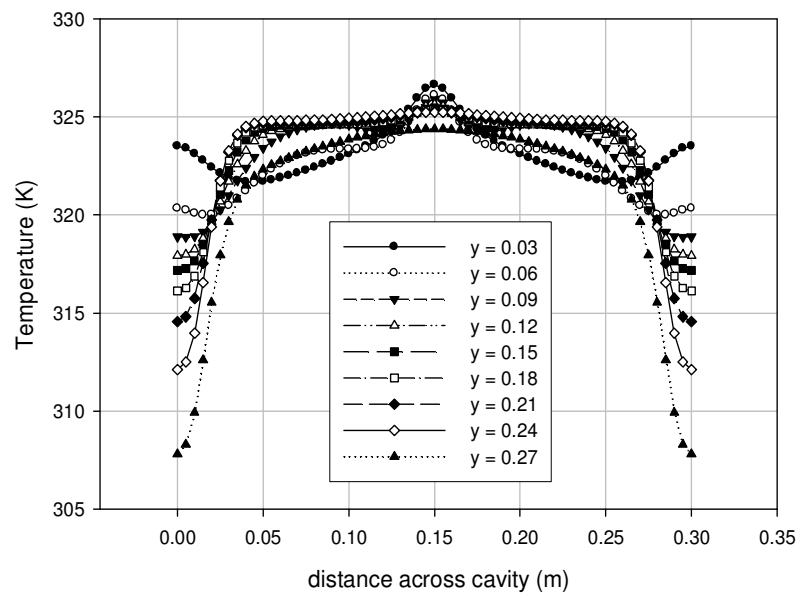


Figure 7.9: *Temperatures at various horizontal lines across a 60 x 60 cell cavity. Temperatures are plotted at 3 cm increments starting at the bottom of the cavity.*

Difference in CHAD and Fluent
Calculated Temperatures Across a Heated Cavity
for Various Axial Levels

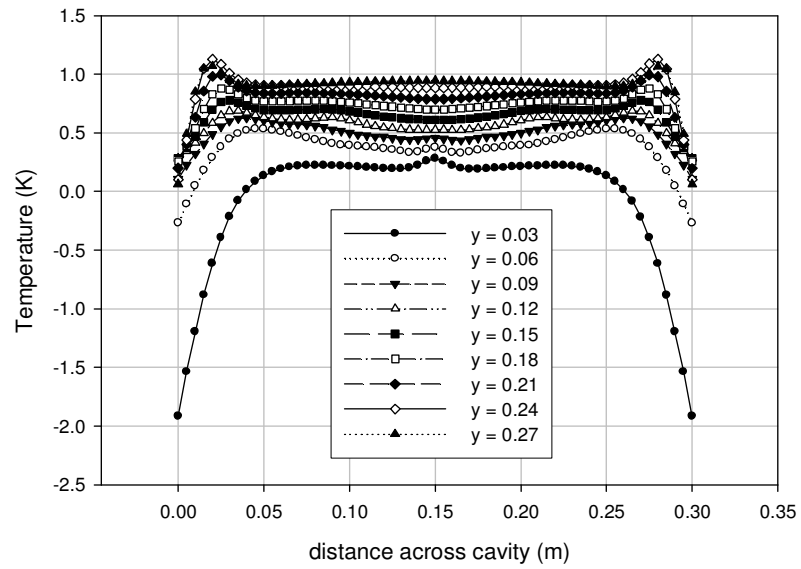


Figure 7.10: *Difference in temperature across a naturally circulating box. The bottom third of the box has a uniform heat source of 10 W/m^3 , while the top wall of the box is fixed at 300 K .*

comparison line, in which case it noticeably under predicts the temperature. The temperature difference is indicative of an error of approximately 5% between the two codes. It is likely that this error is partially the result of slightly different source term distribution between the two models, since the CHAD source term resides slightly higher in the geometry. The remainder of the difference appears to be the result of slightly different flow patterns between the two models.

The differences between CHAD and Fluent for this test problem are relatively small, and demonstrate that CHAD and Fluent can produce results that are similar, at least for this class of problem. In the next chapter, substantially larger calculations are performed using CHAD using some of the features discussed thus far, as well as using Fluent, in an attempt to solve a practical nuclear engineering application.

Chapter 8:

A Practical Nuclear Application Solved Using CHAD & PETSc

8.1 A Practical, Nuclear Design Application

The development of a new reactor design provides numerous opportunities for computational fluid dynamics in design as well as safety analysis (Yadigaroglu et al., 2003, Kwon et al., 2003). As mentioned earlier in this dissertation, the computational requirements of many CFD applications in the nuclear industry are large, and currently untenable for most users. Even the detailed modeling of a fuel channel presses the limits of typical computing clusters beyond the realm of typical engineering analysis. As such, typical engineering applications of CFD are found in smaller, more localized domains. One such application of CFD may lie in the analysis of a proposed control rod drive cooling (CRD) system for BWRs.

The control rod drive systems of BWRs differ from their PWR counterparts because BWRs contain steam separator and steam dryer assemblies above the core. These assemblies restrict the amount of free space that is available to control rod drives that utilize gravity as a fail safe mechanism. Therefore, BWR control rod drives are located below the reactor cores and are driven up into the core region when reactivity control is desired.

Early control rod drives were driven through a complex hydraulic system. This hydraulic system provided only coarse control over the positioning of the control rods

(and thus over the control blades mounted at their tips.) Such control rod drives are typically found in early generation BWRs. One of the requirements of these drives is that the drive internals must maintain temperatures lower than that of the reactor vessel. This prevents damage to certain temperature sensitive interior components within the drive assembly. To maintain drive temperatures below that of the reactor vessel, a slow stream of water is continuously passed through a thin channel (a thermal sleeve) and emptied into the reactor vessel.

To demonstrate CHAD's applicability to a relatively large scale nuclear problem, the flow field in the control rod drive cooling system is determined. A model that is similar to the geometry of a control rod drive's thermal sleeve was developed. The problem is solved using CHAD with and without the PETSc library. Results are obtained for several different mesh densities. The results of these calculations are compared against results obtained using Fluent for a range of increasingly finer meshes. The size of the finest mesh was dictated by the available computing resources on ANL's *Reserve* computing cluster. As in previous calculations, Fluent was used to obtain a reference solution using a machine at the University of Illinois.

A model of a control rod drive cooling system is constructed using the approximate dimensions of an actual control rod drive section. A feed line of approximately 1 inch in diameter provides flow to a semi-annulus at the bottom of the drive mechanism. The feed line is modeled by a rectangle to facilitate a quality, hexagonal mesh that is usable by both Fluent and CHAD. The hydraulic diameter of this

rectangular line is approximately the same as that of an actual circular feed line. The drawback in using circular cross section feed lines would be that the mesh generator could create meshes with elements having a number of faces not allowed by CHAD (such as five) making the mesh generation process more complicated. At the end of the lower annulus, the flow is channeled into two tubes that rise up through the interior of the drive mechanism, emptying into a larger annulus at the top of the drive mechanism.

The annulus at the top of the drive mechanism drains out along all the exterior walls down a thin sleeve which itself drains back into a central annulus where it is collected and returned back into the feed line. Inlet and outlet lines in an actual model would be separate but they are joined here for convenience. Figure 8.1 displays the general layout of the model, while Figure 8.2 shows the actual model that was developed using GAMBIT.

Models were built using three different mesh densities. In the first model, ~225,000 nodes were used. In the second model, the lower annulus and the inlet/outlet tube regions were slightly refined and the nodal count increased to ~250,000 nodes. Finally, in the third model, all regions of the model were refined and ~750,000 nodes were used. Figure 8.3 shows the mesh distribution used for selected regions of the model.

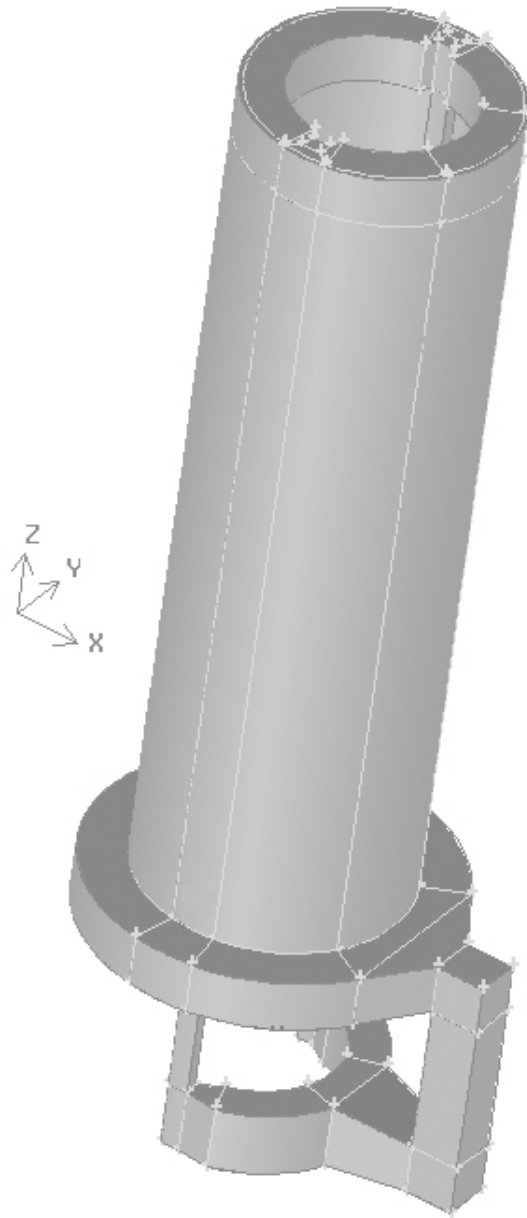


Figure 8.1: A simplified control rod drive geometry. The large cylinder is a very thin sleeve that is fed from the top by two narrow channels that rise up through small tubes inside the shell formed by the sleeve. The mesh geometry is structured, and the light colored lines differentiate the blocks used to create the mesh.

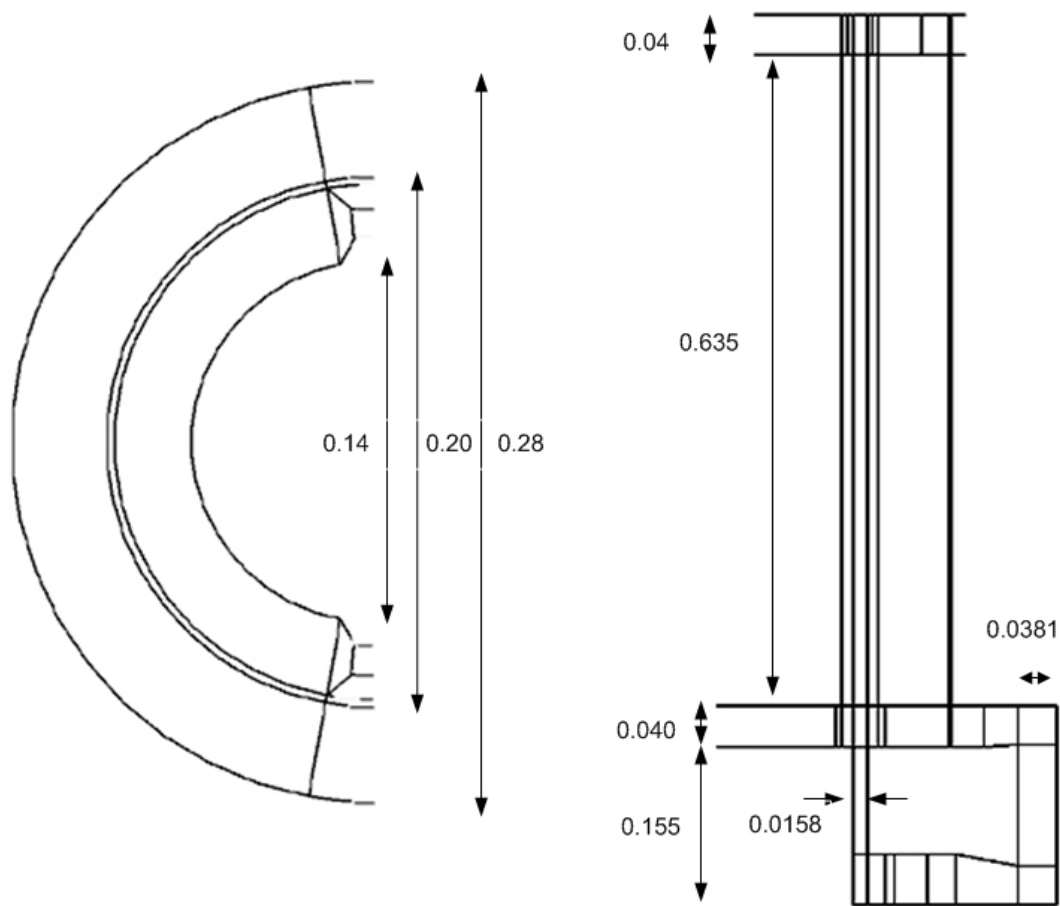


Figure 8.2: Dimensions of the control rod drive model. All units are in meters.

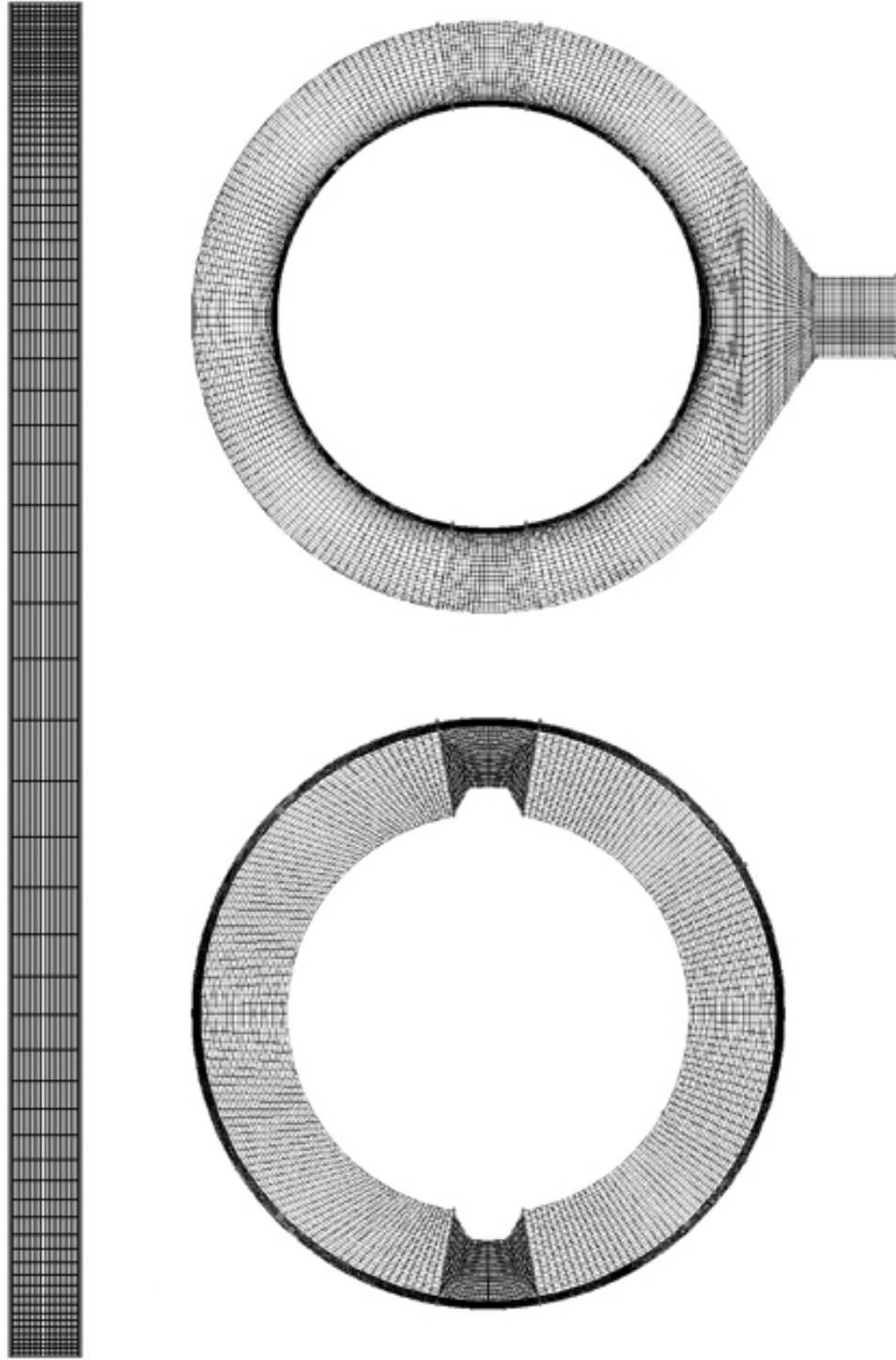


Figure 8.3: Mesh for thermal sleeve, middle annulus, and upper annulus of the control rod drive model. The mesh shown resulted in 742,238 nodes and 652,114 cells.

All external facing boundaries are walls in both Fluent and CHAD. In CHAD, they were set using a `nodetype` of 31 (wall node) with all of the `CONTAB` values set to zero.

The center block in the inlet/outlet tube region has a uniformly distributed volumetric momentum source term of 25 N/m^3 in the negative z direction (downward). This source term provides for a volumetric flow rate on the order of several gallons per minute. The flow in the negative direction remains consistent with other rod cooling designs, and provides cooler water to the more temperature sensitive regions of the drive.

For the low mesh density problems, Fluent and CHAD calculations were performed for both compressible and incompressible fluids. Fluid properties are for an ideal gas at a pressure of 86115 Pa, and with a viscosity of $1 \times 10^{-5} \text{ kg/m-s}$. In CHAD, the same properties were used with a perfect gas. Both CHAD and Fluent used an incompressible fluid having material properties equivalent to the compressible fluids. Since the flow is highly subsonic, the compressibility effect is negligible. For the 750k node calculation, both Fluent and CHAD used their respective compressible fluids. These calculations used these equations of state because it was generally easier to obtain higher levels of convergence in a reduced amount of time.

Both CHAD and Fluent were run until steady solutions were obtained. The CHAD model having 225k nodes and using the incompressible equation of state was solved using PETSc as its continuity solver to demonstrate that the combination of

PETSc and the incompressible fluid could generate the same solution as would be obtained using the compressible fluid and CHAD's native solver. In CHAD, solutions to the 750k node problem using its native solver and a perfect equation of state were obtained in approximately 3 days using up to 16 processors. The combination of the large computation time, and the iteration controls described earlier in this dissertation were the justification for solving the largest CHAD problem using its native solver and a compressible fluid. The same calculation required approximately one day using one processor using Fluent.

8.2 CRD Model Results

CHAD and Fluent simulations were run until a steady state condition was achieved. , The flow fields at selected locations were then compared. Beginning at the central, and largest section of the drive mechanism, Figure 8.4 shows the w component of velocity across the main thermal sleeve of the drive system at a model elevation of $z = +0.39$ m. Here, it is evident that the flow is laminar and well developed. The two low mesh density (225k node) solutions obtained using CHAD agree quite well with each other, despite the use of different equations of state and two different solvers. This is because of the low mach number of the flow. In the case of the more refined mesh, the results obtained using CHAD also agree quite well with the results obtained using Fluent. The difference in the peak velocity magnitude between the refined mesh calculation and the coarser mesh calculations is slightly less than 6%.

The flow from thermal sleeve collects at the top, inner portion of the central annulus region, which is shown as a contour plot in Figure 8.5. The plot, taken from the high mesh density CHAD results, is made at approximately the z -centerline of the annulus. As with the thermal sleeve, the flow is symmetric about the horizontal axis. The blue circle on the right side of the annulus is stagnant flow that results from the symmetry and the location's proximity to the wall. To the right of the stagnant flow, as the fluid moves to the right and flows down (into the page) into the pipe towards the lower annulus, the average velocity of the flow increases as the flow area decreases.

The central portion X - Z plane in the pipe connecting the upper annulus to the lower annulus is shown in Figure 8.6. Though Figure 8.5 shows that the maximum u -component of velocity to be approximately 17 cm/s, Figure 8.6 clearly shows that as the flow turns the corner it substantially increases in maximum speed, increasing to almost 25 cm/s. This is in part due to a further constriction of the cross-sectional flow area.

Because this region is more complicated, and the dominant flow direction changes from $+u$ to $-w$, the w -component of velocity is plotted. This is shown in Figure 8.7. Again the agreement between CHAD and Fluent is relatively good. The right side of the tube contains the bulk of the downward flow that results from the flow coming across the top of the tube at a higher velocity, and then turning down when it reaches the wall. This keeps most of the flow on that side. The left side of the tube is subject to an upwards recirculating flow that is caused by the high u -velocity at the top left corner of the pipe in Figure 8.6. The

CRD Thermal Sleeve Fluid Velocity

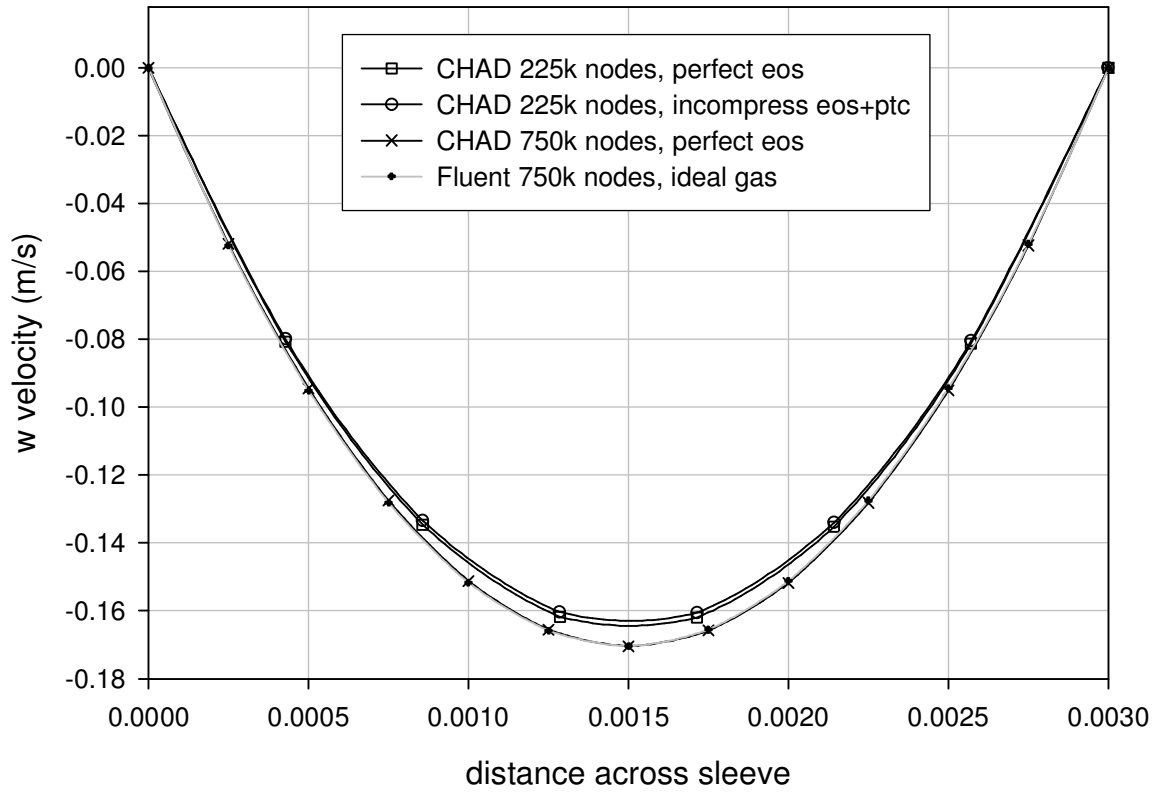


Figure 8.4: Flow across the thermal sleeve at an elevation of $z = 0.39$ m. The agreement between the CHAD solutions is good, though as a group they are slightly smaller in magnitude than those obtained using Fluent.

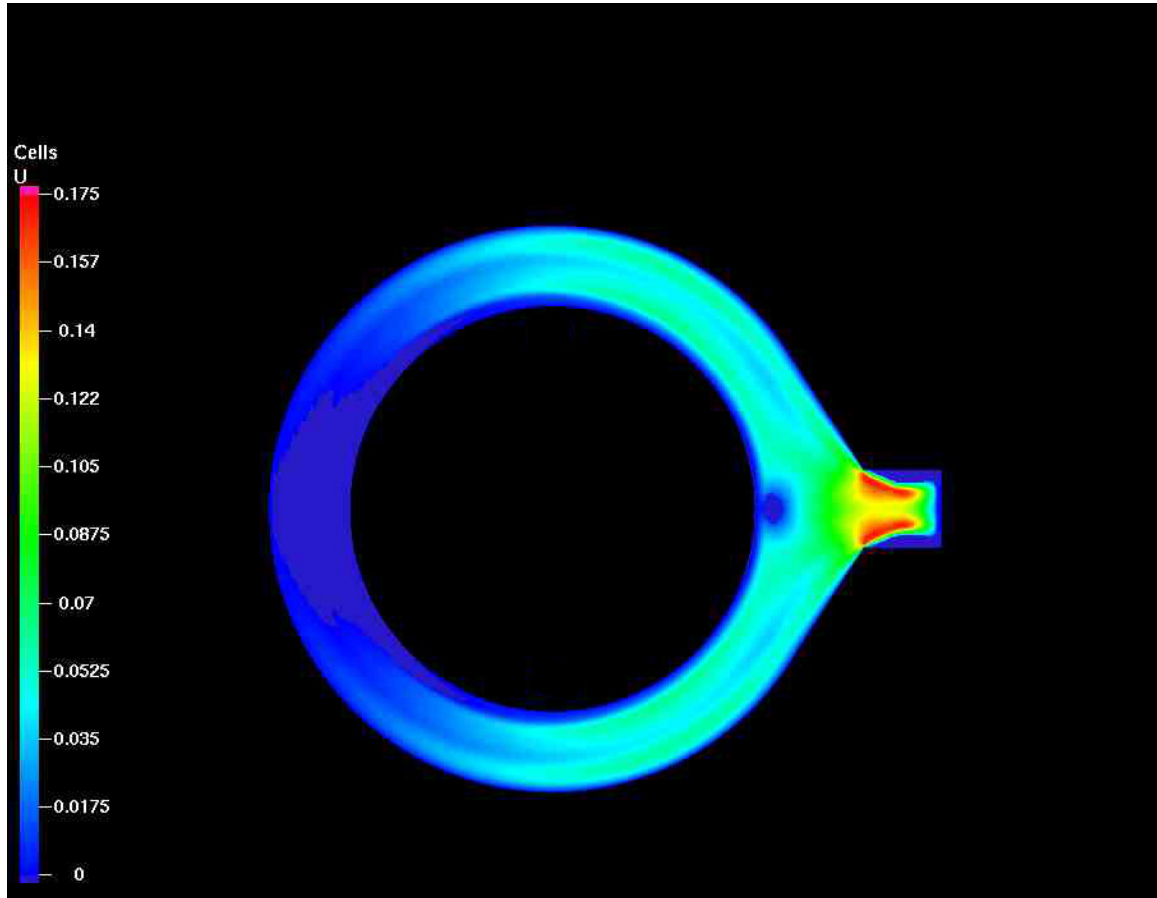


Figure 8.5: The flow field in the center annulus in the X-Y plane. The positive x direction is towards the right of the picture.

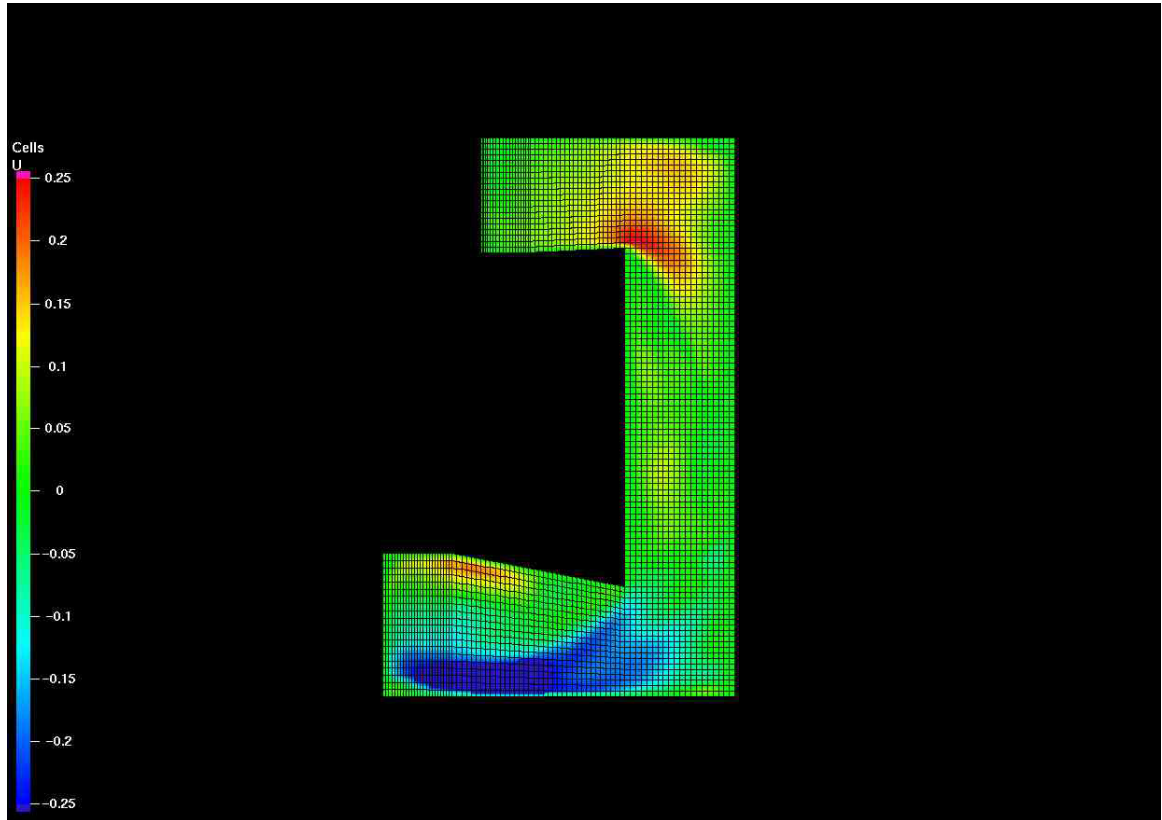


Figure 8.6: U velocity contour in a side view in the X - Z plane of the region with the source term. The change in the u -velocity component is nicely captured. The mesh is overlaid for reference.

CRD Source Region $z = -0.02$

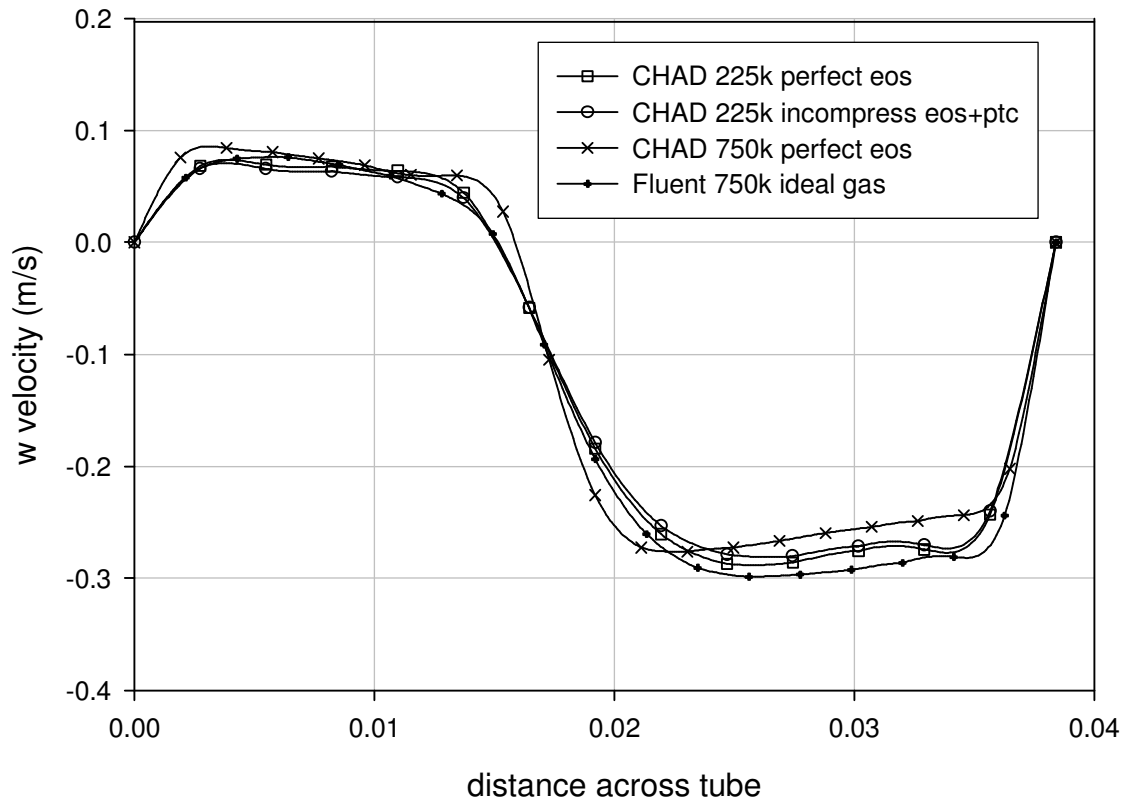


Figure 8.7: A cross section of the w component of velocity across the top of the source region shows good agreement between both CHAD models and between CHAD and Fluent.

Fluent results produce the largest magnitude down flow, while the high density CHAD results produce the lowest magnitude down flow, which is likely due to the slightly wider profile of the downward flow evident in that CHAD solution. The PETSc based CHAD solution again agrees quite well with the normal CHAD solver for the same mesh density.

At the lower annulus, the u -component of velocity is plotted at $x = 0.075$ m and $z = -0.135$ m along a line from $y = -0.2$ to 0.24 m in Figure 8.8. This is a complicated region of the model, having the largest hydraulic diameter, and two diverging flow paths that strongly test the symmetry of the model. Here, the flow field entering the region is sensitive to local details in geometric modeling and meshing as it turns the bottom corner of the pipe. The sensitivity to meshing is evidenced by the jaggedness displayed in the high mesh density CHAD solution. This jaggedness is an artifact of the post processing in which multiple nodes that lie within a specified tolerance along the line along which the data was extracted contribute to the value plotted. Because the postprocessor picks up multiple data points, the difference between two nearby data points is easily seen in the form of sharp local jumps.

Both Fluent and CHAD calculations show some degree of asymmetry between the $+y$ and $-y$ portions of the model. This can be seen by comparing the left side of Figure 8.8 to the right side, for each of the codes. All of the models display the same qualitative characteristics. However, both of the low mesh density CHAD solutions are again in reasonable agreement.

CRD Lower Annulus

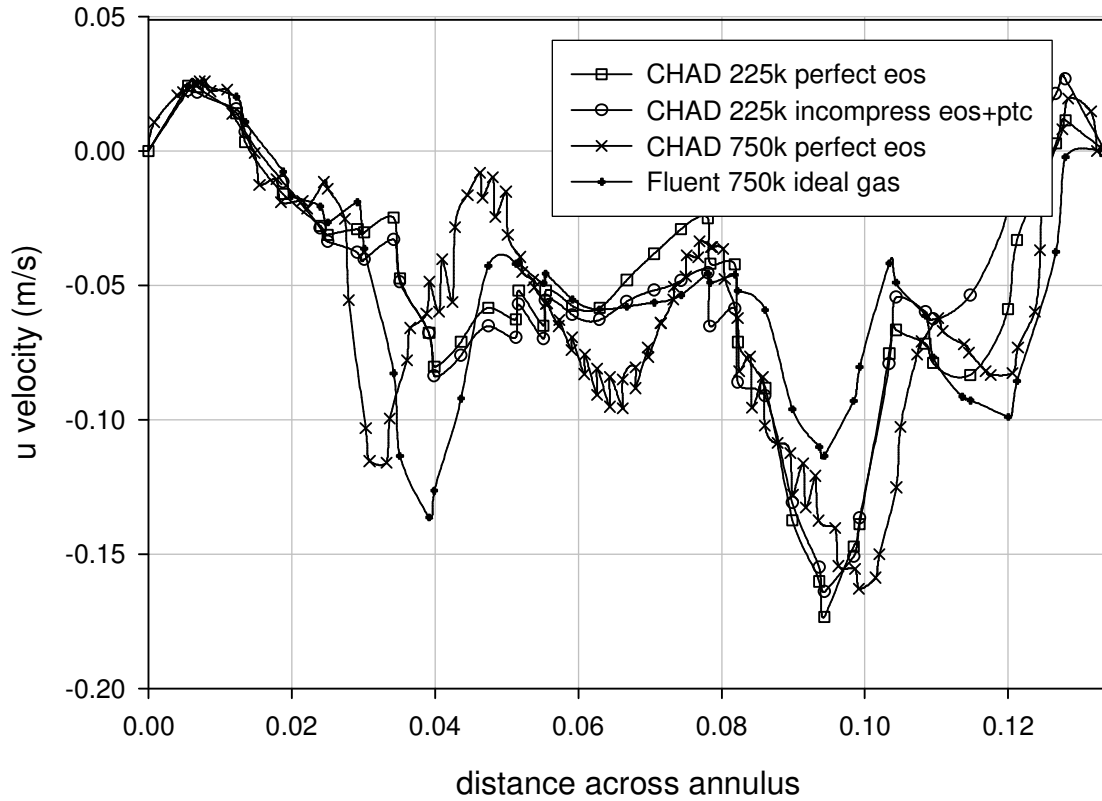


Figure 8.8: The u -component of velocity is plotted across the lower annulus at $z = -0.135$ m. The $+y$ direction is toward the right hand side of the plot. The jaggedness in the 750k CHAD solution is an artifact of the post processing software. It indicates the difference in velocity between neighboring nodes in this region.

This asymmetry may be attributed to a poor quality solution in this region. The difficulty in obtaining a quality solution in this region of the model is almost certainly a result of the flow conditions. In this area, the local Reynolds number is approximately 3,000, which is on the edge of the transition to turbulent flow. As the pipe opens up into the annulus, the increase in hydraulic diameter pushes the flow conditions well into the transition region, making local steady-state solutions very difficult to obtain.

Figure 8.9 shows a contour plot of a selected plane through the center of the lower annulus. From this figure it is easier to see the asymmetry in the lower annulus. In this CHAD solution, the top left and bottom left portions of the geometry narrow before leading to two narrow tubes which rise through the center of the model to the top annulus. As is indicated by Figure 8.8, similar asymmetries can be seen in the results obtained using Fluent. The contour plane at the same axial location but from the high mesh density Fluent results is shown in Figure 8.10. It shows the same qualitative profile as in the CHAD results, and is consistent with the quantitative results shown in Figure 8.8.

Clearly, the conditions in the geometry which had been favorable for a “good quality” solution up until the lower annulus are no longer favorable due to the asymmetry in the solution. This asymmetry now influences the downstream flows. Because the fluid profiles in the positive-y and negative-y regions are different in the lower annulus below the feed tubes, the flow in the feed tubes is also affected. Figure 8.11 shows the CHAD solution for

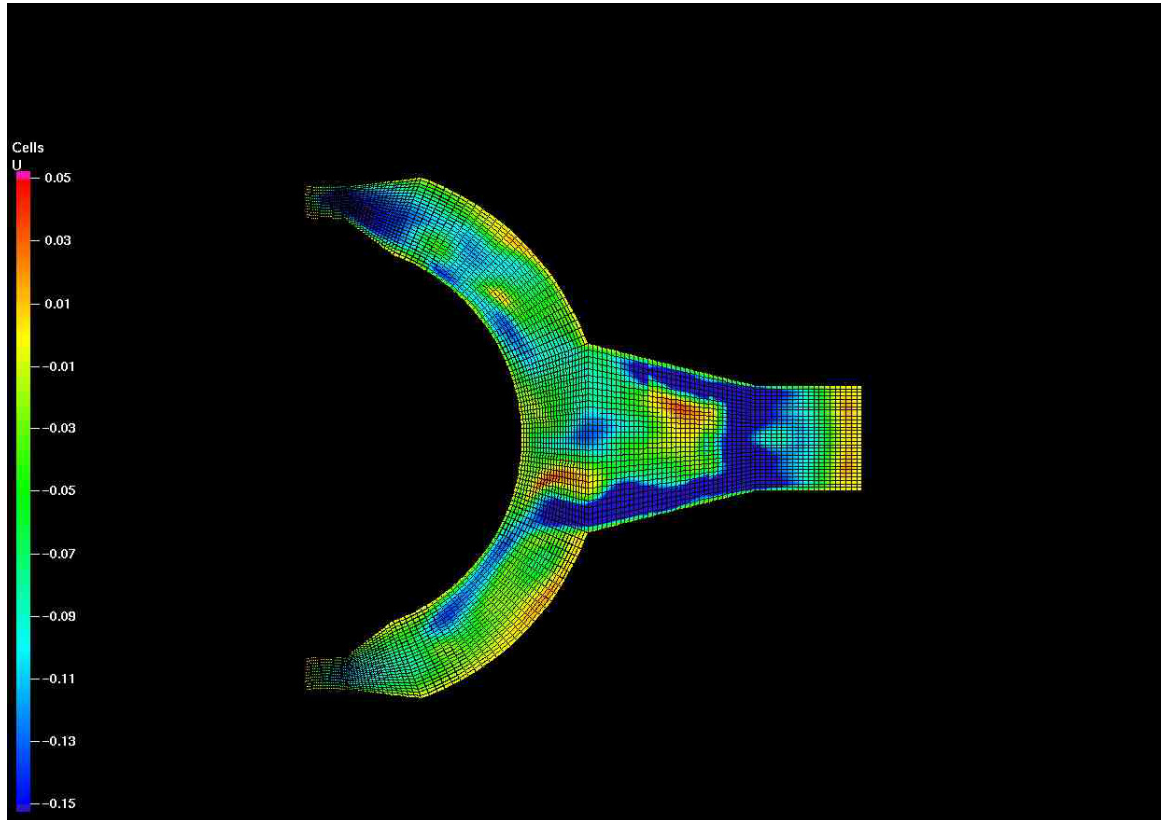


Figure 8.9: The flow field (u velocity, CHAD results) in the lower annulus in the X-Y plane. Note that it is not symmetric. This leads to an asymmetric solution in the feed tubes as well. The mesh is overlaid for reference.

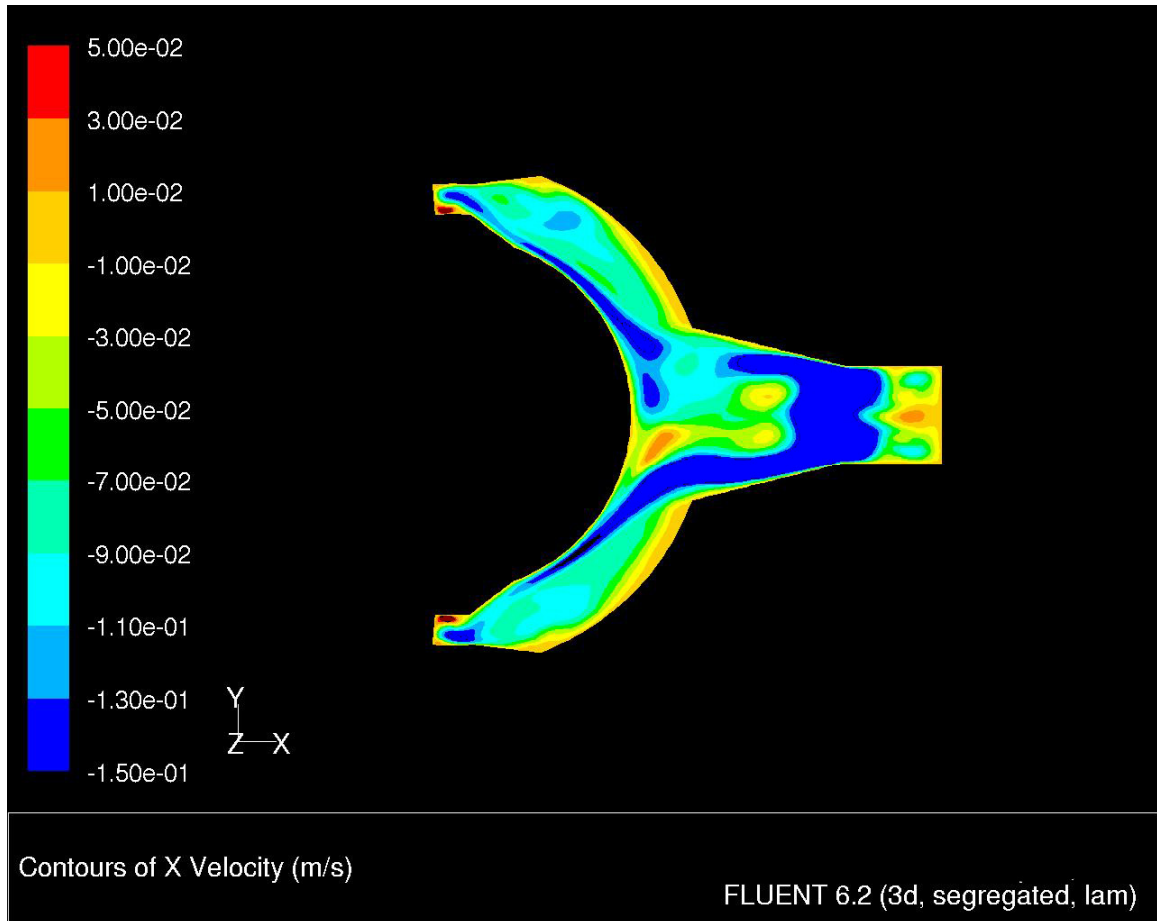


Figure 8.10: The flow field in the lower annulus in the X-Y plane (high mesh density Fluent results). Like the CHAD solution, it is asymmetric. The mesh is the same as the mesh used for CHAD results shown in Figure 8.9.

the velocity in the z direction in the feed tube halfway up the drive mechanism. The maximum velocity in the two tubes is different for both CHAD and Fluent. In Figure 8.11 the difference in maximum upward velocity for the two tubes is approximately 25%. In the Fluent results, the difference in maximum upward velocity is on the order of approximately 10%. The better agreement in the Fluent results is due to the better agreement at the bottom of the feed tube, which acts much like an inlet boundary condition on a pipe.

At the top of the feed tubes, the flow exits into the upper annulus. Because of the difference in feed tube flows, the flow profile at the top is also expected to be asymmetric. Figure 8.12 shows the contours obtained using CHAD at approximately the middle of the upper annulus. The flow around the feed tube ports is symmetric about the X-axis and asymmetric about the Y-axis. This is entirely due to the larger amount of incoming flow in the high-Y portion of the annulus. At this point in the flow path, the fluid moves to the outside edge of the geometry and begins its downward descent into the thermal sleeve. Due to the very thin nature of this sleeve, the flow field rapidly develops into laminar flow and assumes the profile seen in Figure 8.4.

8.3 Discussion of the CRD Model Results

The simulation of the control rod drive model and the inspection of some data at several locations within, offers some insight into the ability of CHAD to be used in nuclear applications. Starting with the assumption that Fluent is a state-of-the-art, commercial CFD

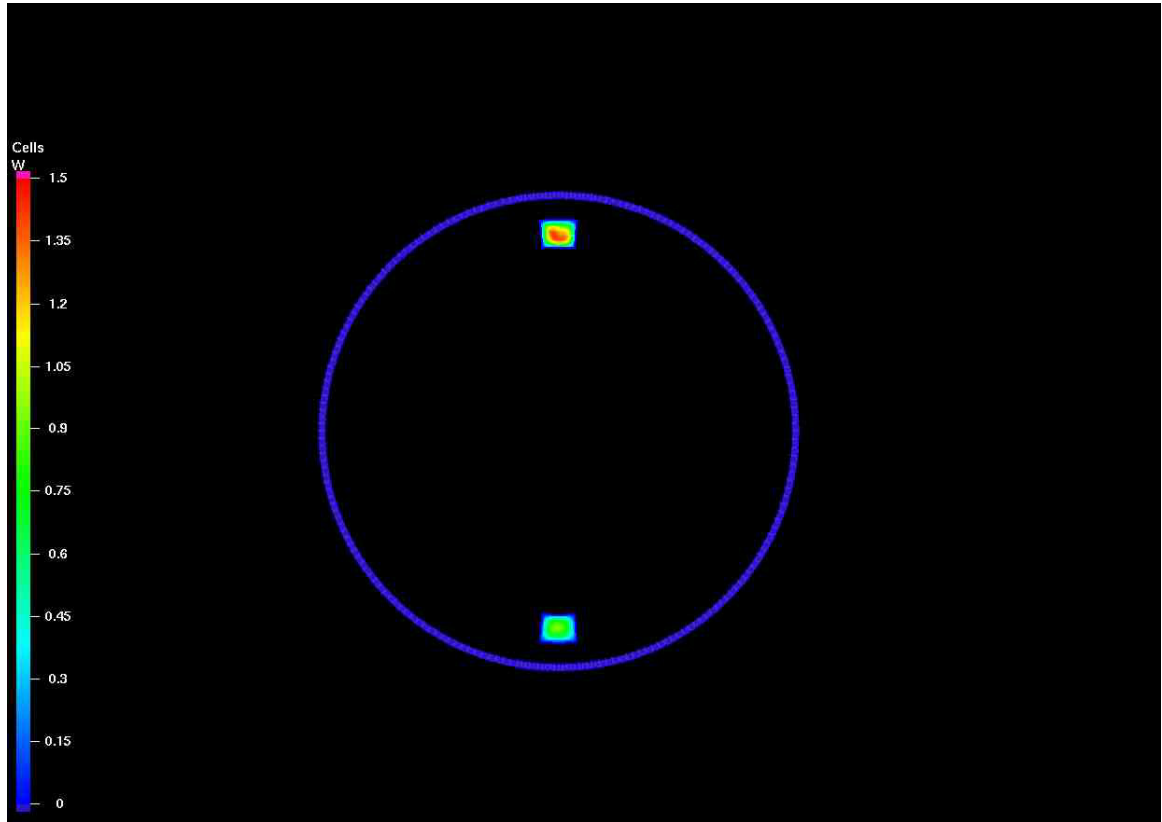


Figure 8.11: Contour of velocity in the z direction halfway to the top annulus. The feed tubes (two small squares) remain asymmetric. A region of high velocity can be seen in the top tube.

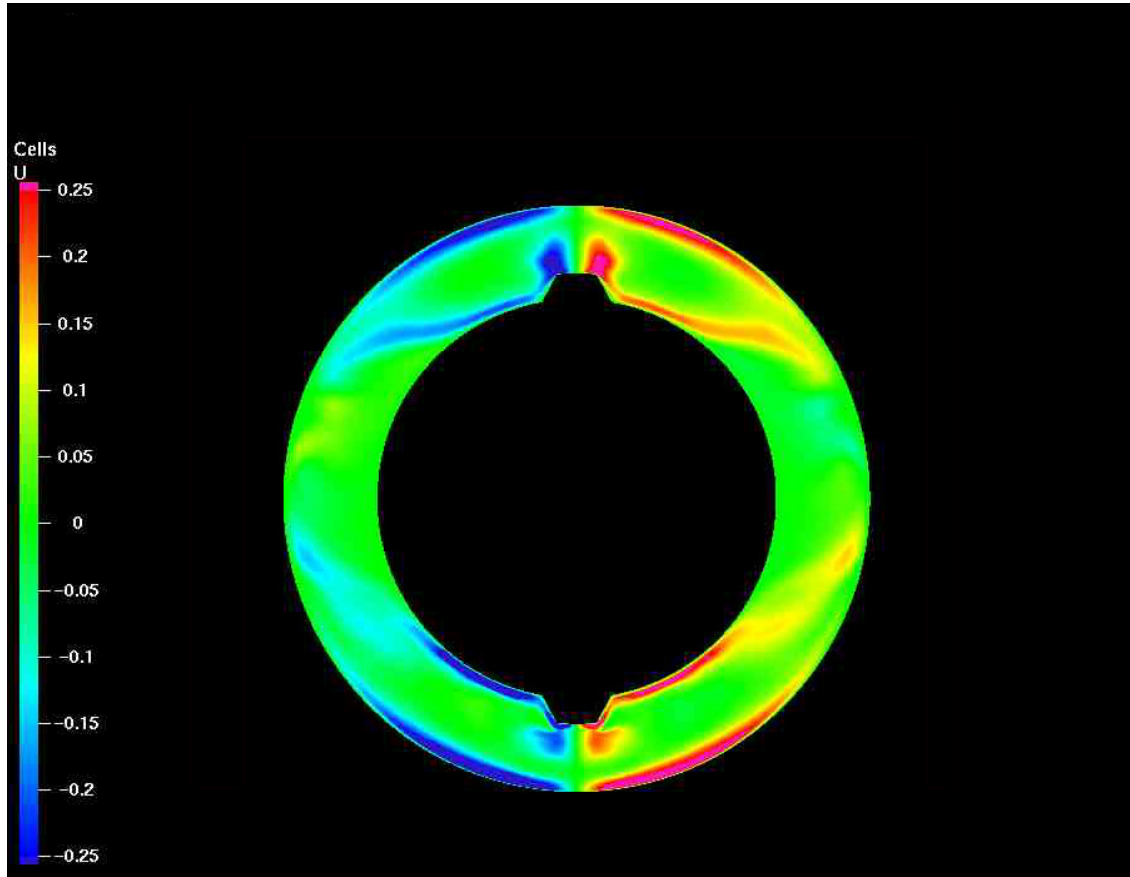


Figure 8.12: Flow field (u velocity, CHAD results) at the top annulus. The only asymmetry appears to result from the difference in the flow coming up through the feed tubes. The remainder of the flow field appears relatively symmetric.

software, and CHAD is either unproven or its capabilities unknown to the general public, the following observations are made.

First, the computation time required to solve the largest of the CRD models using CHAD was significantly higher than Fluent. This was observed when using CHAD's native solver, as well as when using the Pressure Gradient Scaling method. This is an effect that has been seen throughout the work performed in this dissertation.

Second, the comparison between Fluent and CHAD results was good until the geometry and fluid conditions of the model combined to create a situation in which the fluid transitions to turbulent flow. There was good agreement in the thermal sleeve. There was also a nice, symmetric flow field in the central annulus region of CHAD. And the vertical flow profile in the source region was in good agreement between Fluent and CHAD. In the bottom of the source region pipe where the flow enters the lower annulus, the geometry opens up and the flow conditions change. In this location, the flow transitions to turbulent flow. Here, the differences between CHAD solutions and Fluent solutions are more pronounced. There was still some similarity in the flow profiles at this location, however. But beyond this location a comparison between Fluent and CHAD becomes somewhat arbitrary due to the large difference in the flow in the lower annulus between the two codes. The flow in the lower annulus at the entrance to the feed tubes determines the flow through the feed tubes. The flow in these feed tubes differs by an amount dictated by their different inlet conditions, and exit into the top annulus from where they exit with a reasonable profile.

Third, the comparison between CHAD solutions obtained using CHAD's native solver and equation of state agreed well in all cases to those obtained using PETSc and an incompressible equation of state. However, improvements in solution times were not realized due to the convergence criteria and the tolerance settings of the PETSc solver. Therefore, these results provide additional confirmation on the functionality of the solver kit, and acknowledge that further improvement may be made.

Fourth, and finally, a post-calculation inspection of the mesh used in the Fluent and CHAD models led to a potential source of the asymmetry in the solution at the lower annulus. At the +y side feed tube, a mesh seed along one wall in a geometry block had its mesh seed ratio inverted in comparison to the other seeds on the block. This led to a different mesh pattern than was found near the other feed tube. The offending mesh is displayed in Figure 8.13. Good meshing practices generally state that the mesh size in the direction of the flow should increase gradually. Typically, a 30-50% increment over the previous cell size is recommended. In the mesh used for this simulation, this guideline is broken. The solution can be adversely affected, and poor convergence can therefore occur in this location. However, the blame for the asymmetry in the lower annulus may not entirely lie with the poor mesh geometry in this one location. Fluent did not exhibit such a degree of asymmetry, which suggests that perhaps Fluent's solver is not nearly as susceptible to mesh cell size changes. Should this indeed be a reflection on CHAD, it would indicate that great care be taken in the generation of meshed geometries.

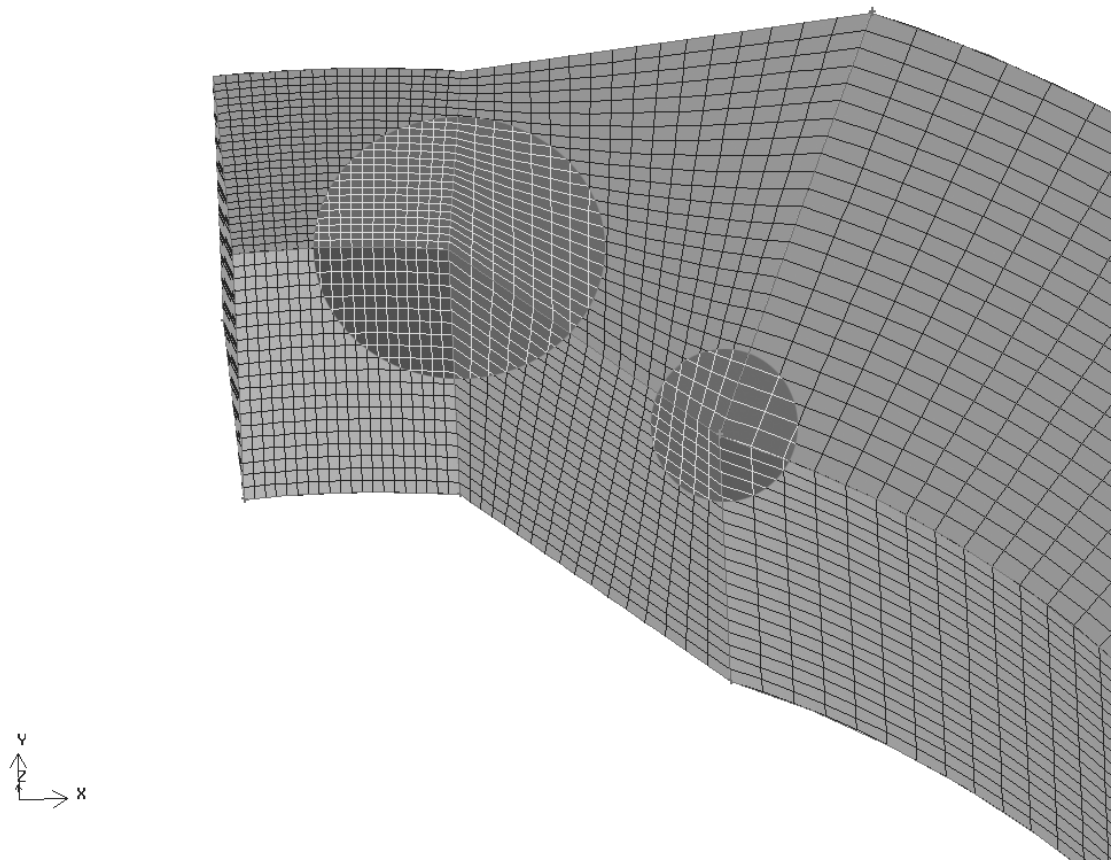


Figure 8.13: The mesh at the +Y feed tube has an inverted mesh seed (show in inverted colors). This may affect the flow patterns in the lower annulus.

Chapter 9: Conclusions and Future Work

9.1 Conclusions

The intent of this dissertation was to start with the CHAD computer code and advance it in some ways that will make it more useful as a tool for the nuclear industry and academia. The primary focus was on adding the PETSc Toolkit to the code and using it to improve the performance of the continuity equation. Other goals were to improve the usability of the code by providing the ability to generate mesh files from a commercial mesh package, and to provide a solution monitoring capabilities that did not previously exist in the code. Another important goal was to demonstrate the applicability of CHAD to solving relatively large, nuclear specific applications. Finally, as CHAD lacks sufficient documentation, a User's Manual was written that serves as a quick start guide for future CHAD users.

At the onset of this dissertation, work previously performed on CHAD at Argonne National Laboratory was migrated into a version of CHAD two generations newer. This work included both a reformulated continuity equation and a first order upwinding scheme. Both the reformulated continuity equation and the upwinding schemes were tested against a benchmark from literature to validate the migration process.

To provide a tool to help measure convergence, two tools were developed and incorporated into CHAD. The first tool allows the user to monitor the residual of selected variables at a coordinate within the domain of the problem. The second tool

measures the absolute normalized residuals of the momentum equations, energy and mass. Both of these tools may be activated from the input file.

The PETSc Toolkit was then added to CHAD. It was added as a complimentary solver to both the reformulated continuity equation, and a modified form of CHAD's original continuity equation. The PETSc Toolkit can be activated from the command line along with a set of parameters which can select solvers and preconditioners, or adjust the control of the solution, including the convergence criteria. For selected test cases, the PETSc solutions, both for CHAD's native continuity equation as well as the reformulated one, were found to be nearly identical to those obtained when CHAD's native solvers were used. Additionally, these solutions could be obtained in a reduced amount of CPU time. However, the improved performances were found to be somewhat problem dependent. When time step sizes were held constant, the PETSc solver typically outperformed CHAD's native GCR solver, however, when time step sizes are allowed to vary, the GCR solver tended to outperform PETSc's solvers when semi-implicit calculations were performed. This is because the algorithm used to determine time step growth in the semi-implicit solver partially depends on the number of time steps used to reach convergence during the solution of the continuity equation. Since PETSc's matrix-based solvers do not directly allow for CHAD's convergence to be inferred during each iteration, they are not able to participate usefully in the time step size adjustment.

The CHAD/PETSc combination was tested to determine the influence of the *relative tolerance* measure of convergence. The relative tolerance measures the degree to

which PETSc solves its matrix system of equations. Relative tolerances of between 0.1 and 0.01 were found to be optimal for the variety of problems solved in this dissertation. Even for problems not discussed within this dissertation, values smaller than 0.01 were rarely ever found to be advantageous. Values of r_{tol} smaller than 0.01 tended to lead to overly accurate intermediate convergence which is not useful in CHAD's SIMPLE like algorithm. Values larger than 0.1 produced overly coarse solutions that did not adequately help the continuity equation converge, and therefore resulted in an increased number of outer iterations, thereby increasing the overall number of iterations in the hydrodynamic solution. This increased number of iterations leads to an increase in total computation time. Therefore, relative tolerances larger than this are not recommended.

The *restart* factor for the PETSc solver GMRES was also tested. By saving information from previous iterations, the solution algorithm can achieve convergence at an increased rate at the cost of increased memory usage. This factor was found to be minimally important for the problems tested.

A variety of solvers and preconditioners were used to measure the performance of them on a sample test problem. GMRES is PETSc's default solver and was one of the best performing solvers. GMRES performed comparably to the Conjugate Residual solver when paired with the same preconditioners as CR. On four processors, GMRES outperformed CR by an average of about 15% total CPU time.

An incompressible equation of state was added to CHAD. The preconditioner for the GCR solver was modified so that it would not be subjected to division by zero errors due to incompressibility, and it was modified to be able to be used with PETSc. Several two-dimensional tests cases were run using buoyant flow models to demonstrate the equation of state and the use of PETSc within CHAD. One of the test cases was taken from the literature, and the results of the comparison agreed well. Another test case was designed from scratch and compared against solutions obtained using the commercial CFD code Fluent. Despite differences between the two software codes, the agreement between them was reasonable.

A mesh refinement study was performed using CHAD to measure the convergence rate for one of the test cases. The code was found to converge by the power of 2.58, indicating greater than second order convergence. The expected convergence rate was at least 2.0.

To assist in the generation of a useful test problem, a mesh generation utility was developed that utilizes Fluent's Gambit preprocessor and converts a neutral mesh file into a CHAD readable format. This utility allows identical meshes to be developed and used between CHAD and Fluent.

Using the newly developed mesh generation utility, the largest problem attempted within this dissertation was created and solved. It involved simulating the hydraulic flow inside a conjectured control rod drive mechanism for a boiling water reactor. This drive

mechanism channels flow up through the center of a short cylinder where it is collected at the top, and flows down the outside of the cylinder. There, it is again collected and pumped down and then up through the center of the cylinder again. Typical flow speeds for this kind of application are quite slow, and the problem is essentially laminar throughout. This geometry was meshed with three different mesh densities and solved using CHAD with its original solvers, CHAD with PETSc and an incompressible equation of state, and Fluent. As a highly successful commercial CFD code, a high density Fluent calculation is used as a reference solution.

The mesh with the smallest number of nodes, approximately 225,000, was solved using all of the Fluent and CHAD configurations. The CHAD configurations, compressible and incompressible, both agreed very well with each other, again providing confidence in the implementation of PETSc. Both configurations also showed mixed agreement in other regions of the model. In the most developed areas, the agreement with Fluent was good, but in some of the larger areas, the CHAD configurations showed some asymmetric behavior. Refining the mesh in some troubled areas, increasing the overall mesh density to 250,000 nodes, improved the solution somewhat. The highest density CHAD solution, equivalent in all regards to that used by Fluent for its high density calculation, improved the agreement between CHAD and Fluent substantially, though some regions still showed some disagreement in the most complicated region of the model. This region of the model was found to have Reynolds numbers well into the laminar-turbulent transition region. Flow fields with Reynolds numbers in the transition region are difficult to solve. These calculations display results in this part of the model

that indicate that both CHAD and Fluent had difficulty. The downstream flow fields in both Fluent and CHAD were also somewhat affected by this.

A post-calculation inspection of the mesh shows a slight imperfection near one of the feed tubes up the center of the geometry. This could be a source of some of the asymmetry seen in the CHAD solution, though, if this is indeed the cause of some variation in CHAD's results, it would indicate CHAD is more sensitive than Fluent in this regard.

The CHAD solutions, whether they used PETSc or not, took substantially longer to obtain than those obtained by Fluent. This may be in part to assumptions or simplifications that Fluent makes (such as whether or not it is using a structured grid), but evidence seems to indicate that the actual problem may lie within the communication routines. This could be located anywhere from within the code to within the communication library being used, or within the MPICH implementation.

Because CHAD is a poorly documented piece of software, indeed there is no user's manual and only a draft version of a physics manual, a User's Manual designed to assist new users of CHAD with issues from installation to code inspection to execution of the code was written. Additional information about CHAD has also been provided with this dissertation, such as information regarding boundary conditions, both at corner nodes and via the CONTAB array, parallel communication, and the mesh translation utility.

9.2 Future Work

If in the future others decide to pursue using CHAD for academic or commercial applications, there is much that can be done to advance it. One important thing that should be done is a number of test cases that are relevant and typical to the nuclear industry should be constructed and solved using CHAD and a variety of other CFD codes (and if possible compared against experimental data.) The test problems that come with the CHAD distribution tend to be focused around shock type applications. Having test cases on hand with reference solutions would provide the industry with a head start on gaining confidence in and using CHAD. Solving these test problems will also show the code functions as intended. Additional test cases will also provide confidence in aspects of CHAD not explored in this dissertation. For example, in the experience of this author, the *k-epsilon* model in CHAD did not perform as expected. However, it would be premature to say whether this could be a bug in the code or whether it is a lack of sufficient testing. Having additional test cases would bring any such issues to attention

Another area of action would be to take an in depth look at boundary conditions within CHAD. Certainly, there is elegance in the way in which the boundary conditions that lie at the intersection of walls and inflow or outflow boundaries has been implemented. However, it may not necessarily be the way that CHAD users wish to handle them. In particular, it would simplify things if the faces at those corners were not closed. The application of boundary conditions is another area for improvement. Currently, providing anything but the simplest boundary conditions to CHAD requires

writing user implemented FORTRAN. This must be built into the CHAD executable. Having an automated way for which CHAD can read boundary condition files would substantially minimize the time required to construct different models. Also, because it is anticipated that boundary condition changes will be made in future versions of CHAD, CHAD was not modified to generate PETSc solutions involving wall nodes that lie at the intersection of inflow and outflow boundaries.

The solution speed of CHAD also requires examination. Currently, CHAD requires significantly more time to solve a given problem than does other CFD codes. This should be found to be substantially reduced. In some sense, CHAD's versatility is its own enemy. For example, being a fully unstructured code, it assumes everything is unstructured and pays the performance hit accordingly. But this may not be the only thing killing CHAD's performance. The communication associated with CHAD is strongly suspect. In the cases performed on the *Reserve* cluster at Argonne, the vast majority of the time can be attributed to the communication routines. Our experience has been that even when communication has been optimized, such as using a Metis partitioned grid, performance has not improved substantially. It is possible that such performance might be attributed to the configuration of MPICH or PGSLib on *Reserve*, but when CHAD has been run on other Linux machines at UIUC a similar communication behavior has been observed.

The solution speed of the PETSc solver can also be improved by optimizing the time step size control for the semi-implicit solver. Currently, the time step size is a

function of the number of iterations required to achieve convergence in the continuity equation. The time step size adjustment might be altered to choose a new time step size in a different way, or, possibly more preferably, the number of iterations for which PETSc takes to achieve convergence can be better controlled. The measure of convergence in the continuity equation could be adjusted so it can be monitored by PETSc, for example, by performing ten iterations, then checking against CHAD's convergence criteria, then iterating again. This author experimented with dynamically altering the value of r_{tol} but without much success.

Post processing of CHAD can be improved as well. It would be rather useful if CHAD could output data into a hierarchical data format, such as HDF5 (HDF Group). This would enable the use of rapid post processing capabilities currently freely available.

References

Amsden, A.A., O'Rourke, P.J., and Butler, T.D., "*KIVA-II: A Computer Program for Chemically Reactive Flows with Sprays*," Los Alamos National Laboratory report LA-11560-MS, May, 1989.

ANSTO, Summary of the Preliminary Safety Analysis Report (PSAR) for the ANSTO Replacement Research Reactor Facility," www.ansto.gov.au, 2001.

Anglart, H., Nylund, O., Kurul, N., and Podowski, M.Z., "*CFD Prediction of Flow and Phase Distribution in Fuel Assemblies with Spacers*," Nuclear Engineering and Design, **177**, pp. 215-228, 1997.

Armstrong, R., Gannon, D., Geist, A., Keahey, K., Kohn, S., McInnes, L., Parker, S., and Smolinski, B., "*Toward a Common Component Architecture for High-Performance Scientific Computing*," LLNL, UCRL-JC-134475, 1999.

Balay, Satish, Gropp, William D., McInnes, Lois C., and Smith, Barry F., "*PETSc Users Manual*," Argonne National Laboratory, ANL-95/11 - Revision 2.1.0, 2000.

Cambridge Power Computing Associates, *PGSLib User Guide*, Cambridge Power Computing Associates Ltd., 1997

[a] Canfield, Thomas R., Chien, Tai-Hsin, Domanus, Henry M., Tentner, Adrian M., Tzanos, Constantine P., Valentin, Richard A., Weber, D.P., “*A Coupled Newton-Krylov Solver for Improved CHAD Cache Utilization and Performance*,” ANL, ANL-MCS-P800-0300, 2000.

[b] Canfield, Thomas R., *personal communication –email*, Sept. 29, 2003.

[c] Canfield, Thomas R., *personal communication –email*, Jan. 30, 2003.

Chun, Tae-Hyun, and OH, Don-Seok, “*A Pressure Drop Model for Spacer Grids With and Without Flow Mixing Vanes*,” Nuclear Science and Technology, Vol 35. No. 7, pp. 508-510, July 1998.

Collins, J.P., Collela, P., and Glaz, H.M., “*An Implicit-Explicit Eulerian Gudonov Scheme for Compressible Flow*,” Journal of Computational Physics, **116**, 195, 1995.

Computational Engineering International (CEI), *Enight User Manual for Version 8.0*, 2005.

CUBIT Development Team, *CUBIT Mesh Generation Environment Volume 1: Users Manual*, Sandia National Laboratories, SAND94-1100, April 18, 2000.

de Vahl Davis, G., “*Natural Convection of Air in a Square Cavity: A Benchmark Numerical Solution*,” International Journal for Numerical Methods in Fluids, Vol. 3, pp 249-264, 1983.

de Vahl Davis, G. and Jones, I. P., “*Natural Convection in a Square Cavity: A Comparison Exercise*,” International Journal for Numerical Methods in Fluids, Vol. 3, pp 227-248, 1983.

Elman, Howard C., *Iterative Methods for Large, Sparse, Nonsymmetric Systems of Linear Equations*, Research Report #229, Yale Univ., New Haven CT (thesis reprint), pp.30-53, April, 1982.

Ferziger, J. H. and Peric, M., *Computational Methods for Fluid Dynamics*, Springer-Verlag Berlin, Heidelberg, New York, 2002.

Fluent Inc., *Gambit 1.3 Documentation*, January 2005.

Fox, Robert W., and McDonald, Alan T., *Introduction to Fluid Dynamics*, John Wiley & Sons, 1992.

Frank, G., and D’Elia, J., “*CFD Modelling of the Flow Around the Ahmed Vehicle Model*,” Proceedings of the 2nd Conference on Advances and Applications of GiD, 2004.

Ghia, U., Ghia, K. N., and Shin, C.T., “*High-Re Solutions for Incompressible Flow Using the Navier-Stokes Equations and a Multigrid Method*,” Journal of Computational Physics **48**, pp. 387-411, 1982.

Gnu Compiler Collection Homepage, <http://www.gnu.org/software/gcc/gcc.html>

Greenbaum, Anne, *Iterative Methods for Solving Linear Systems*, Society for Applied and Industrial Mathematics, 1997.

Griebel, Michael, Koster, Frank, Meyer, Michael and Croce, Roberto, “*Documentation for Software Package NaSt3DGP*,” Institut für Angewandte Mathematic Der Universität Bonn.

Gropp, William, Lusk, Ewing, and Skjellum, Anthony, “*A High-Performance, Portable Implementation of the {MPI} Message Passing Interface Standard*,” Journal of Parallel Computing, Vol 22, no. 6, pp. 789-828, Sept. 1996.

Gropp William D., Dinesh K. Kaushik, David E. Keyes, and Barry F. Smith, “*High Performance Parallel Implicit CFD*”, Journal of Parallel Computing **27**, pp. 337-362, 2001.

Gropp, William, Lusk, Ewing, and Skjellum, Anthony, *Using MPI – Portable Parallel Programming with the Message-Passing Interface*, 2nd Ed. The MIT Press, Cambridge Mass., 1999.

Grunwald, G., Höhne, Kliem, S., Prasser, H.-M., Rohde, U., and Wiess, F.-P., “*Experiments and CFD Calculations on Coolant Mixing in PWR – Application to Boron Dilution Transient Analysis*,” Technical Meeting on the use of CFD codes for Safety Analysis of Reactor Systems, including Containment, Pisa, Italy, November 15, 2002.

Ha, J., and Aldemir, Tunc, “*Pool Dynamics of Natural Convection-Cooled Research Reactors*,” Nuclear Technology, **79**, pp 297-310, 1987.

Haworth, D.C., and Jansen, K., “*LES on Unstructured Deforming Meshes: Towards Reciprocating IC Engines*,” Center for Turbulence Research, Proceedings of the Summer Program, pp. 329-346, 1996.

HDF Group, The, www.hdfgroup.org.

Hestenes, M.R., and Stiefel, E., “*Methods of Conjugate Gradients for Solving Linear Systems*,” J. Res. Nat. Bur. Standards, **49**, pp 409-435. (1952)

Hoffman, Klaus A. and Chiang, Steve T., “*Computational Fluid Dynamics for Engineers Volume II*,” Engineering Education System, Wichita, KA, 1995.

Keyes, D. E., Kaushik, D. K., Smith, B. F., “*Perspectives for CFD on Petaflop Systems,*”
CFD Review, ed. M. Hafez, World Scientific, Singapore, 1079-1096, 1998.

Knepley, M., Sameh, A. H., and Sarin, V., “*Design of Large Scale Parallel Simulations,*”
Proceedings of Parallel CFD'99, A. Ecer et al., 1999.

LANL, SESAME: The LANL Equations of State Database, Los Alamos National
Laboratory, LA-UR-92-3407, (1992).

Meijerink, J.A., and van der Vorst, H.A., “*An Iterative Solution Method for Linear
Systems of which the Coefficient Matrix is a Symmetric M-Matrix,*” *Math. Comp.*, **31**,
pp.148-162, 1977.

Mohsen Karimian S.A., Straatman, A.G., “*Benchmarking of a 3D, Unstructured, Finite
Volume Code for Incompressible Navier-Stokes Equation on a Cluster of Distributed-
Memory Computers,*”, 19th International Symposium on High Performance Computing
Systems and Applications, pp. 11-16, 2005.

MPICH Home Page, <http://www-unix.mcs.anl.gov/mpi/mpich/>.

Nagayoshi, Takuji, and Nishida, Koji, “*Spacer Effect Model for Subchannel Analysis – Turbulence Intensity Enhancement Due to Spacer Grids*“ Nuclear Science and Technology, Vol 35, No. 6, pp. 399-405, June 1998.

Norris, S. E., *A Parallel Navier-Stokes Solver for Natural Convection and Free Surface Flow*, Thesis, University of Sydney, 2000.

O’Rourke, Peter J. and Sahota, Manjit S., “*A Variable Explicit/Implicit Numerical Method for Calculating Advection on Unstructured Meshes,*” Journal of Computational Physics **143**, pp. 312-345, 1998.

[a] O’Rourke, Peter J. and Sahota, Manjit S., *NO-UTOPIA: The Flow Solver for the CHAD Computer Library*, Physics Manual, LANL, undated.

O’Rourke, Peter J., Saohta, Manjit S., *A Parallel, Unstructured-Mesh Methodology for Device-Scale Combustion Calculations*, LANL, LA-UR-98-4779, 1999.

Ortega, Frank A., *General Mesh Viewer Users Manual*, Los Alamos National Laboratory, LAUR 95-2986, 2004.

[b] Ortega, Frank A., GMV Version Changes,
<ftp://laws.lanl.gov/pub/gmv/Readme.changes> (undated)

Patankar, S. V., *Numerical Heat Transfer and Fluid Flow*, Hemisphere, New York, 1980.

Paterson, J. E., Poremba, L. J., Peltier, S. A., and Hambric, E. G., “A *Physics-Based Simulation Methodology for Predicting Trailing-Edge Singing*,” 25th Symposium on Naval Hydrodynamics, 2004

Peric, M., Kessler, R., and Scheuerer, G., “*Comparison of Finite-Volume Numerical Methods with Staggered and Collocated Grids*,” *Computers and Fluids*, Vol. 16, No.4, pp. 389-403, 1998.

ProDesktop <http://www.ptc.com>

Ramshaw, J.D., O'Rourke, P.J., and Stein, L.R., “*Pressure Gradient Scaling Method for Fluid Flow with Nearly Uniform Pressure*,” *Journal of Computational Physics* **58**, pp 361-376, 1984.

Rhie, C.M., Chow, W.L., “*Numerical Study of the Turbulent Flow past an Airfoil with Trailing Edge Separation*,” *AIAA, J.*, **21**, pp. 1525-1532, 1983.

Saad, Youcef, and Schultz, Martin H., “*GMRES: A Generalized Minimum Residual Algorithm for Solving Nonsymmetric Linear Systems*,” *SIAM Journal of Scientific and Statistical Computing* Vol. 7, No. 3, pp. 856-869, July 1986.

Sahota, Manjit S., “*Programming Conventions in CHAD*,” LANL, LA-UR-99-2734.

Sahota, Manjit S., email to Rizwan-uddin, Feb 7, 2001.

Scheuerer, M., Heitsch, M., Menter, F., et al., “*Evaluation of Computational Fluid Dynamic Methods for Safety Analysis*,” Nuclear Engineering and Design, **235**, pp. 359-368, 2005.

Selmin, V., “*The Node-Centered Finite Volume Approach: Bridge Between Finite Differences and Finite Elements*,” Computer Methods in Applied Mechanics and Engineering, **102**, pp 107-138, 1993.

Strohmaier, Erich, Dongarra, Jack J., Meuer, Hans W., Simon, Horst D., “*Recent Trends in the Marketplace of High Performance Computing*,” Parallel Computing, **31**, pp. 261-273, 2005.

Thode, L.E., Cline, M.C., DeVolder, B.G., Sahota, M. S., Zerkle, D.K., “*Comparison Among Five Hydrodynamic Codes with a Diverging-Converging Nozzle Experiment*,” LANL, LA-13653, 1999.

Trefethen, L.N., Bau III., D., *Numerical Linear Algebra*, SIAM, Philadelphia, 1997.

[a] Tzanos, Constantine P., "*Development of a Two-Phase Flow CHAD Capability – Homogeneous Equilibrium Model*," Internal memo, Argonne National Lab, undated.

[b] Tzanos, Constantine P., "*Reformulation of CHAD's Continuity Equation*," Internal memo, Argonne National Lab, Dec. 16, 1998.

[c] Tzanos, Constantine P., "*CHAD Boundary Conditions*," Internal memo, Argonne National Lab, July 23, 1997.

[d] Tzanos, Constantine P., "*CHAD Boundary Conditions at Corner and Edge Nodes*," Internal Memo, Argonne National Lab, Feb. 4, 1998.

[e] Tzanos, Constantine P., "*Effect of Boundary Conditions on CHAD Predictions*," Internal memo, Argonne National Lab, Feb. 19, 1998.

[f] Tzanos, Constantine P., "*CHAD Bottlenecks*," Internal memo, Argonne National Lab, Oct. 26, 1999.

[g] Tzanos, Constantine P., "*Central-Difference-Like Approximation for the Solution of the Convection-Diffusion Equation*," Numerical. Heat Transfer, Part B, **17**, pp. 97-112, 1990.

Van Leer, B., “*Towards the Ultimate Conservation Difference Scheme. Part V. A Second-Order Sequel to Gudonov’s Method,*” Journal of Computational Physics, **32**, 101, 1979.

Vinsome, P.K.W., “*Orthomin, An Iterative Method for Solving Sparse Sets of Simultaneous Linear Equations,*” Proc. 4th Symposium on Reservoir Simulation, Society of Petroleum Engineers of AIME, 149-159, 1976.

Weber, D.P, Wei, T.Y.C., Brewster, R.A., Rock, D.T., and Rizwan-uddin, “*High Fidelity Thermal-Hydraulic Analysis Using CFD and Massively Parallel Computers,*” Proceedings of the 4th International Meeting on Supercomputing Applications in Nuclear Engineering, Japan, Sept. 2000.

Yadigaroglu, G., Andreani, M., Dreier, J. and Coddington, P., “*Trends and Needs in experimentation and Numerical Simulation for LWR Safety,*” Nuclear Engineering and Design, **221**, 205-233, 2003.

Yan, Y., Rizwan-uddin, and Sobh, N., “*CFD Simulation of a Research Reactor,*” American Nuclear Society Proceedings of the Mathematics and Computation, Supercomputing, Reactor Physics and Nuclear and Biological Applications, Avignon, France, 2005.

Appendix A: Communication in CHAD

A.1 Introduction

There are three primary types of mesh structures in CHAD: nodes, connections, and elements. In this dissertation, we have discussed only nodes and connections. In this appendix, we discuss nodes and connections in greater detail than was given in the body of the main chapters.

A.2 Nodes

Nodes are essentially points in space where variables such as u , v , w , p etc. are stored. In CHAD, we denote nodes by v , not to be confused with the similar velocity term. Data for nodes is stored in 1-D arrays where the size of the array is equal to the number of nodes. In multiprocessor simulations, the size of these arrays is equal to the number of nodes residing on that processor. Nodes are assigned local, unique id numbers on the processor they reside on. Nodes also have a unique global id number assigned to them.



Figure A.1: Nodal notation

When nodes are part of a structured mesh, a general relationship exists among adjacent nodes. No such relationship exists between nodes in unstructured meshes. CHAD assumes the use of unstructured meshes at all times. With no information available regarding adjacent nodes, another method must be used to relate nodes.

A.3 Connections

In CHAD, connections are used to relate adjacent nodes. Connections are denoted using α . Like nodes, each connection has a unique identification number on each processor. For each connection, a 2-D array is used to keep track of node ids at each end

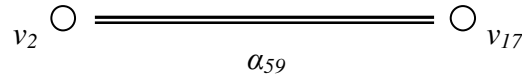
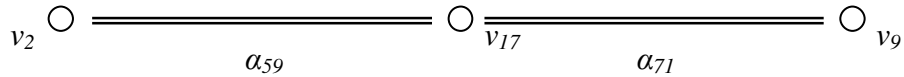


Figure A.2: A connection and two nodes at the connection termini

of the connection. In CHAD, this array is called NODEST. The first array dimension is of size two, while the second dimension is of a size equal to the number of connections on that processor. The first entry of the first array dimension for any connection contains the id of the lower numbered node on that connection. The second entry of the first array dimension contains the id of the higher number node.



$$\text{NODEST}(1,59) = 2$$

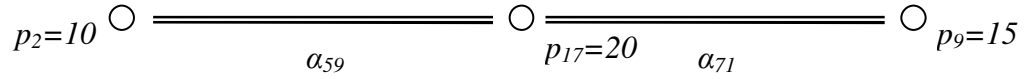
$$\text{NODEST}(2,59) = 17$$

$$\text{NODEST}(1,71) = 9$$

$$\text{NODEST}(2,71) = 17$$

Figure A.3: The node termini array tracks the node identification numbers at either end of a connection. The lower node number always resides in the first array entry.

Similar connection “terminus” arrays exist for most if not all nodal variables. In these arrays, the value of the variable associated with the ends of the connections is stored.



$$PT(1,59) = 10$$

$$PT(2,59) = 20$$

$$PT(1,71) = 15$$

$$PT(2,71) = 20$$

Figure A.4: Variable termini arrays contain the value of the node at the associated connection terminus. This example shows the nodal value of pressure stored in a pressure termini array.

Since connections bridge two adjacent nodes, they span the interface between the control volumes associated with each node. Since it is important to know and retain the

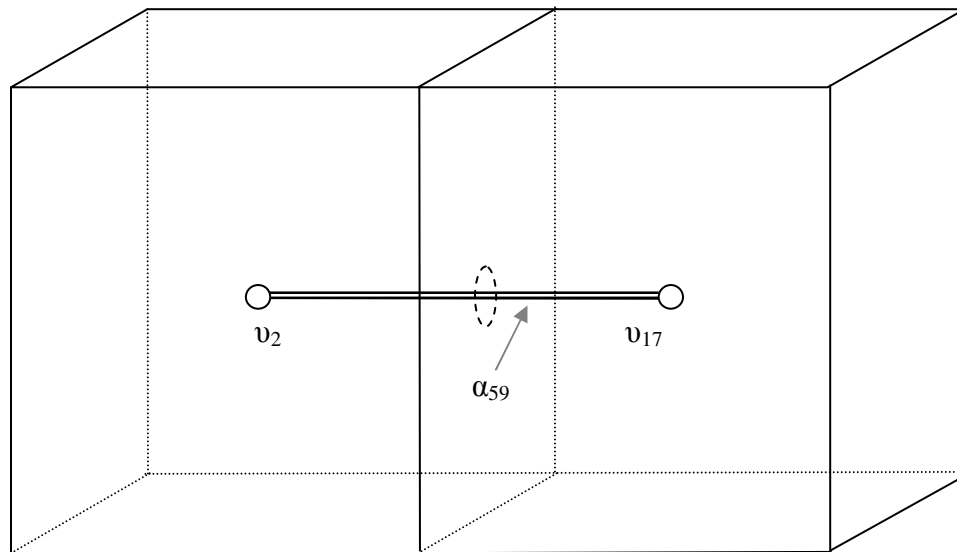


Figure A.5: Connections pierce a cell-face in the normal direction to the cell-face area.

value of nodal values interpolated at the cell-faces, arrays are kept for nodal variables on each connection (u_α , v_α , w_α , p_α etc.). These cell-face (also referred to as median-mesh) variables are stored in 1-D arrays of a size equal to the number of connections.

A.4 Gather Operations

When nodal data is required to compute cell-face values, the terminus arrays associated with connections are used in gathering nodal data for the computation. The gather operation simply uses the NODEST array to look up a node's id and insert the corresponding value of a variable into the terminus array using the node id as the index for the proper location.

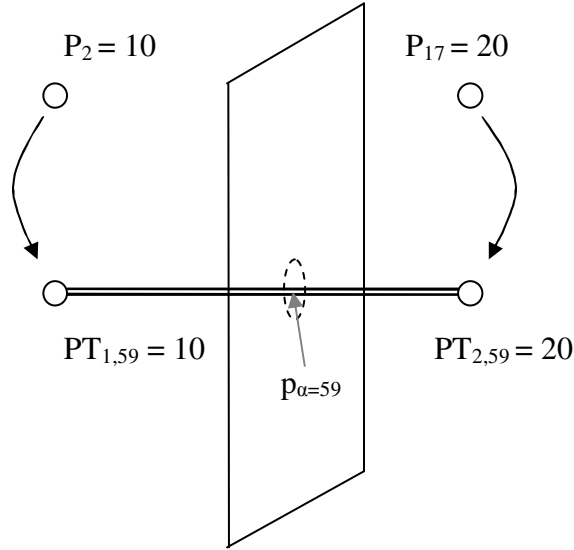


Figure A.6: Gather operations copy nodal variable values to connection terminus arrays.

Similar operations exist for gathering nodal data from connection midpoints to element edges, from nodes to element vertices, and from parent to children nodes. In the example shown in Figure A.6, the nodal values are being copied into a connection termini array. A typical calculation might then be to average the termini values and obtain a cell-face value. The averaging process does not occur in the gather routine, however. It is left up to the user to halve the values before they are gathered to the connection termini.

$$p_{\alpha=59} = \left(\frac{PT_{1,59} + PT_{2,59}}{2} \right)$$

A.5 Scatter Operations

Scatter operations are essentially the opposite of gather operations. As such, scatter operations transfer from element edges to connection midpoints, connection midpoints to nodes, children nodes to parents, and connection termini to nodes. In most of the work covered in this dissertation, scatter operations are used to transfer data from connection termini to nodes, values from terminus arrays are used to update nodal arrays. An example of the use of a scatter operation is the transfer of cell-face mass flow rate residuals onto nodes to determine the error in continuity for a given node.

The error in the mass flow rate across a cell-face affects both nodes equally, but with opposite magnitudes. Typically, the value of the cell-face variable is assigned to one end of a terminus array, and its negative value is copied to second value of the terminus array.

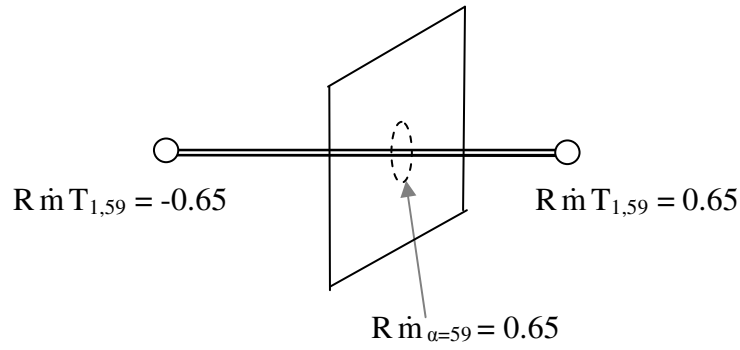


Figure A.7: Scatter operations transfer cell-face variables to nodes

When a node is associated with more than one connection, the termini values may be added into the node, or either the maximum or minimum value of the termini values being assigned to the node can be selected, depending on the arguments provided to the specific scatter function being called. In the residual computations for cell-face mass fluxes, the residuals are added.

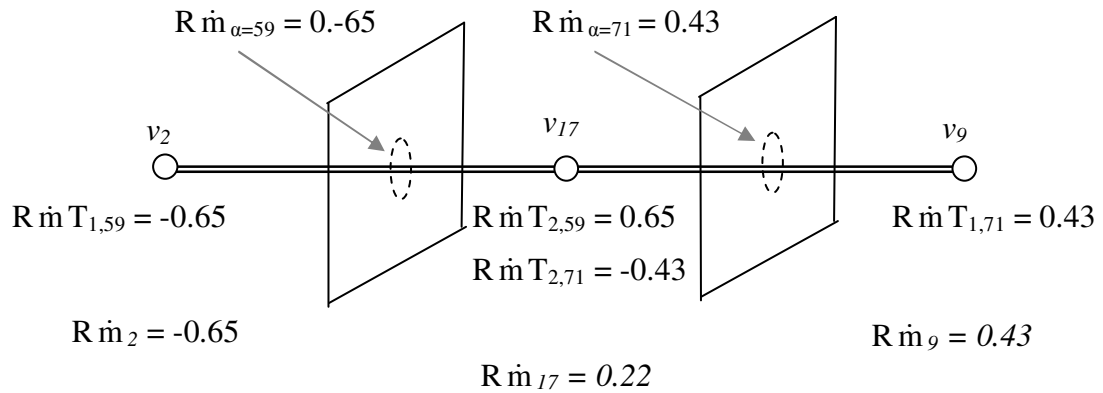


Figure A.8: A more complicated example of a scatter operation.

Appendix B: The Gambit to CHAD Mesh Translator

B.1 Introduction

The CHAD distribution does not include a geometry building program, and it includes only a rudimentary meshing capability (through a separate utility) suitable for the most simple, single block shapes. Therefore, almost all models and meshes destined for use by CHAD must be constructed using other means. Some of the work performed at ANL was done using a simple FORTRAN routine which could construct three-dimensional blocks. This routine is similar to the capability provided through CHAD's utility. Los Alamos National Laboratory uses the commercially available ICEM-CFD software which has the capability of exporting meshes into a CHAD usable format. The first two options listed above are inadequate for building realistic domains of any physical system besides a cavity or duct flows. The latter option requires either current in-house ICEM-CFD capability, or purchasing the software to obtain such a capability. Thus, other methods of obtaining geometries and meshes which are usable in CHAD are desired.

An in-house geometry building and meshing capability currently exists through Fluent Inc.'s Gambit preprocessor. It has the capability to generate the unstructured meshes desired for modeling complex geometries. Additionally, it can export the mesh in a Neutral mesh format (Fluent, 2005). With documentation available for this mesh

format, a translation program was written that converts a Gambit Neutral file into a CHAD mesh file.

This appendix describes the translation program designed to translate Gambit Neutral file format meshes into CHAD readable format. The program was written using Microsoft Visual Basic 6.0 (VB6) and is intended to be used within Microsoft Windows.

B.2 Rules for Generating Meshes Destined for CHAD

The construction of the model geometry and mesh in Gambit must be performed according to a few rules:

- ◆ Gambit should be configured for a *generic* solver. This ensures the proper file format is used upon exporting the mesh
- ◆ Boundary groups should be constructed on the *faces* that will be used to identify the nodes on CHAD boundary conditions. The output option for writing boundary groups should be specified as `NODE/VERTEX` to ensure that nodal boundary condition data is written instead of cell-face boundary data.
- ◆ Wall boundary condition groups must be created after any boundary groups of other types. This is because the `CONTAB` of the wall boundary group must ultimately overwrite the `CONTAB` for inlet/outlet boundaries. At a later date, this restriction may be removed by further development of the translator.
- ◆ CHAD is capable of modeling only tetrahedral, wedge, and hexagonal elements. Therefore, Gambit meshes should not contain elements of other orders.

B.3 Gambit Neutral Mesh Format Translation Procedure

The translation program from the Gambit neutral file format into a CHAD readable format was written using a three step process. Each step is initiated by the user depressing a button on the translator GUI (Figure A.1). In the first step, the neutral file is modified so that is easily read into VB. In the second step, the modified file is then read into CHAD. The third and final step consists of writing the actual CHAD mesh file. Each step is now discussed thoroughly.

B.3.1 Preparing a Gambit Neutral File

The procedure for preparing a Gambit neutral file for translation into CHAD is now given. The program begins by opening the neutral file as selected by the user through a standard Windows Common Dialog box. Next, the translation program reads the neutral file line by line. On each line, it recursively iterates replacing all instances of double blank spaces with instances of single white spaces until each data value is separated by only a single white space. This new single spaced line of text is then written to an output file also selected using a Common Dialog box. Upon writing the last line of text to the output file, both files are closed. At this time, the output file may be inspected by the user.

B.3.2 Running the Translator

The next step of the translator involves opening the single-spaced output file from the previous step. Again, lines of text are read one at a time. On each line that is read, the single white spaces are replaced by commas. This new, comma-delimited line is then

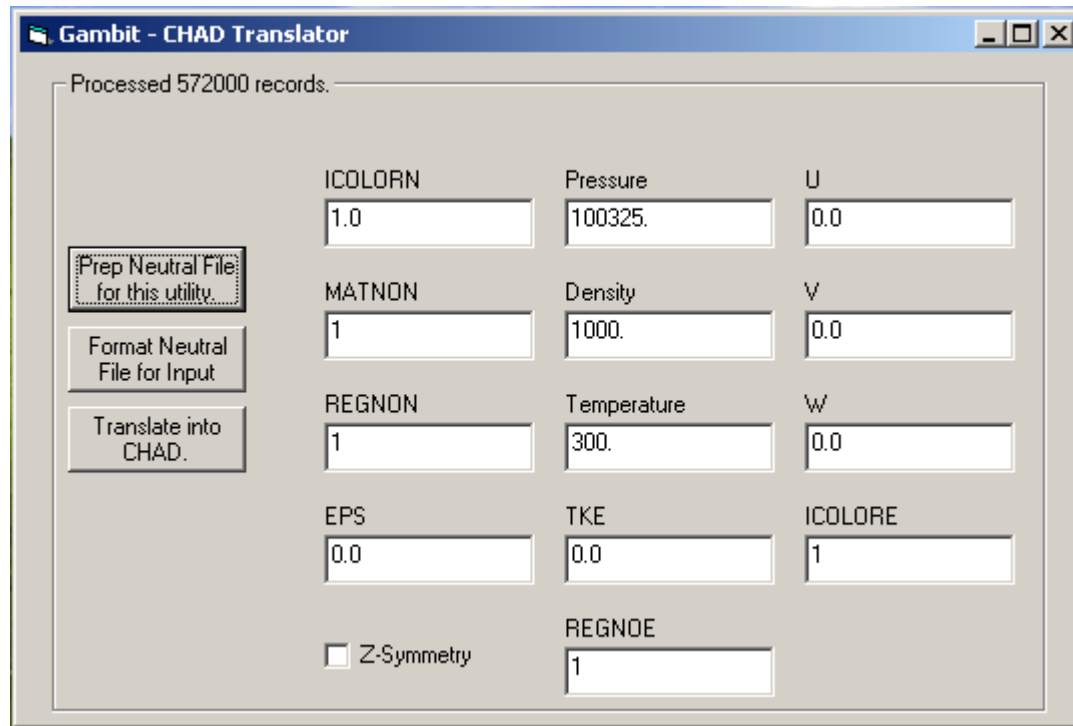


Figure B.1: Gambit to CHAD mesh translator GUI. A CHAD mesh is converted from a Gambit Neutral mesh file format by pressing the three buttons in sequence from top to bottom. Text boxes and a check box set options within the translation process.

written to an output file. Upon reaching the end of the input file for this step, both the input file and the comma-delimited file are closed.

The third major step involves reading the data contained in the comma-delimited file. The structure of the data contained in the file is given in the Gambit neutral file format documentation (Fluent, 2005). A few general statements are now in order.

Integer data is stored using VB Long variables which are 4 bytes in length and span the range of -2,147,483,648 to 2,147,483,647 (Microsoft, 2000). Floating point variables are of stored in VB Double precision covering a range of -1.79769313486232E+308 to -4.94065645841247E-324 for negative values; and 4.94065645841247E-324 to 1.79769313486232E+308 for positive values. Character strings (STRING) are of essentially unlimited length.

The first relevant data read involves the number of nodes, elements, element groups, and boundary condition sets. Using the number nodes and elements, nodal and element based arrays are dynamically sized at run time. Then, nodal coordinate data is read into the arrays for X, Y, and Z.

Next, element data is read. First, the element number is read, followed by the element type, the number of vertices, and finally, the id numbers of the nodes that correspond to the element vertices. If the number of vertices associated with the element does not equal 4, 6, or 8, (corresponding to tetrahedral, wedge, or hexagonal cells) an error message is given and the program aborts.

Gambit neutral files supply only four vertices for tetrahedral cells, and six for wedge cells. CHAD requires that eight vertices be supplied for element data, even when an element contains less than 8 vertices. CHAD collapses excess vertices from hexahedral cells into adjacent vertices to form elements of lower order. This process is illustrated in Figure A.1. For tetrahedral elements, the fourth vertex collapses into the

third, while the sixth, seventh, and eighth vertices collapse into the fifth. The new vertex numbering scheme is then 1,2,3,4. The CHAD vertex list for this element would then contain the following node id sequence: 1,2,3,3,4,4,4,4. A similar method is used for wedge elements, resulting in a sequence of 1,2,3,3,4,5,6,6. The translator uses the reverse of this process to ensure eight vertices are supplied for all cells, regardless of order.

Boundary group data is read next. For each boundary group read, an output file name is generated by combining the name of the boundary condition group with a “.dat”. The node ids found in the boundary condition groups are copied to output files for later, optional use with CHAD.

The translator prepares node type data as its next task. By default, nodes are initially set as interior nodes (`nodetype = 0`), unless the option for symmetry is selected. When the option for symmetry is selected, the default node type is free-slip. `CONTAB` values are initially set to (1,0,1,0,0,1) for all nodes. Boundary condition sets are

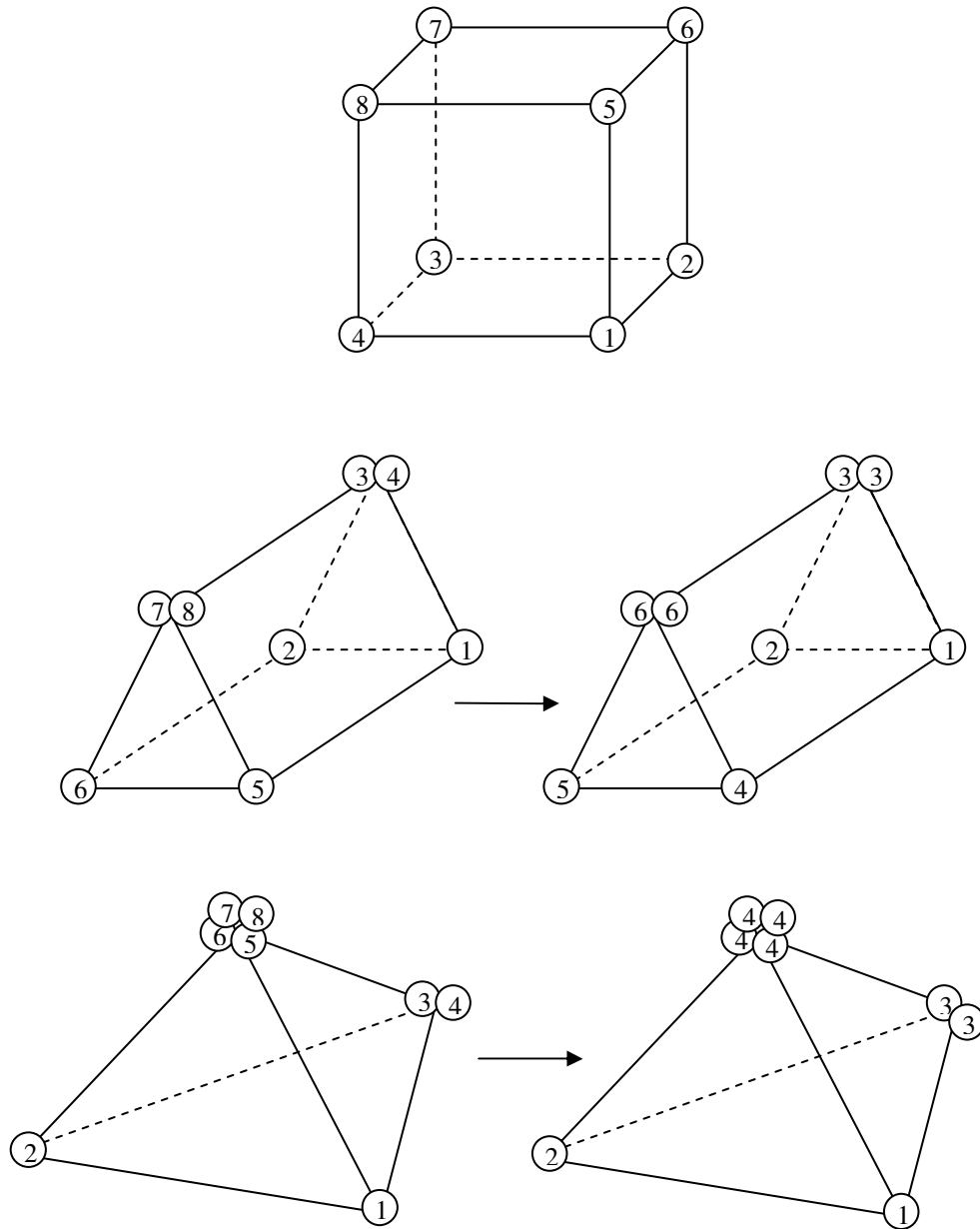


Figure B.2: Eight node numbering scheme for hexahedral, wedge, and tetrahedral control volumes. Wedge and tetrahedral elements require eight vertices to be supplied to CHAD, which is accomplished by repeating node numbers in the vertex list.

then read, with each data value indicating a node id belonging to that set. The name of the boundary condition set is used to determine the node types for the nodes in that set. For names containing 'wall', no-slip nodes are specified. Names containing 'inlet' are set to inlet specified to inflow nodes, while 'outlet' specifies outflow nodes. 'symmet' specifies symmetry nodes. The keywords may be located anywhere within in the boundary set name. Case is sensitive.

After reading in the input file, a Common Dialog prompts the user for a file name and path where the CHAD mesh file data will be written into. On the first line in the file, the number of elements is written twice, followed by the number of nodes, written twice. Finally, three zeroes are written. Nodal coordinate data is written next. All of the nodal X values are written, followed by Y, and then Z position data.

Element vertex data is written next. For each element, eight vertices are provided according to the procedure previously described. Then, various other CHAD input arrays are written (ICOLORN, REGNON, and MATNON) with a default value of one. Next, node types are written for each node, followed by the six CONTAB values for each node. Each CONTAB value is separated by a space. Finally, default values are written for pressure, density, temperature, turbulent kinetic energy and dissipation, u, v, and w components of velocity, and element coloring and region numbers (P, RHO, T, TKE, EPS, U, V, W, ICOLORE, REGNOE).

B.4 The Mesh Translator Source Code

The source code for the main translation subroutines is provided below. Various supporting code, such as that required for button clicks, text box and check box operations and miscellaneous event driven structures are not provided. As execution times for the translator are very short, no effort has been made at optimizing the code. Similarly, as VB is a rapid development language, no effort was made to particularly clarify the code. However, ample comments are included in the source code and should guide the reader in comprehending the structure of the code.

The source code is provided for the following four routines:

1. RemoveSpaces
2. CommaDelimit
3. ReadCommaDelimit
4. WriteCHAD

Module level public variables are declared as follows:

```
Option Explicit

Public sFile As String      ' input file name for preper
Public vfile As String      ' output file for prep/input for translator

Public X() As Double        ' nodal x-coordinate
Public Y() As Double        ' nodal y-coordinate
Public Z() As Double        ' nodal z-coordinate
Public VTX() As Long
Public CONTAB() As Long
Public NODETYPE() As Long
```

```

Public Sub RemoveSpaces()

    ' the purpose of this routine is to read a text file
    ' and reduce it to a single-spaced file.
    ' later we will replace those single-spaces with commas
    ' and perform additional input preparations

    Dim tmpstr As String
    Dim tmpresult As String
    Dim rflag As Boolean
    Dim lngPos As Long
    Dim intEvts As Integer
    Dim lngRcd As Long
    Dim evtCounter As Long

    'common dialog control
    Form1.dlgCommonDialog1.InitDir = App.Path
    Form1.dlgCommonDialog1.Filter = "All Files (*.*)|*.*"
    Form1.dlgCommonDialog1.ShowOpen
    If Len(Form1.dlgCommonDialog1.FileName) = 0 Then
        Exit Sub
    End If

    ' get the input file name
    sFile = Form1.dlgCommonDialog1.FileName

    ' open it
    Open sFile For Input As #1

    ' open the output file
    Form1.dlgCommonDialog1.Filter = "All Files (*.*)|*.*"
    Form1.dlgCommonDialog1.ShowSave

    ' get the output file name
    vfile = Form1.dlgCommonDialog1.FileName

    ' open it
    Open vfile For Output As #2

    ' set the hourglass
    Form1.MousePointer = 11

    ' read every line of the file. It must be in a text format (have
line delimitations)
    Do While Not EOF(1)
        ' read a line (requires line delimitation / text format)
        Line Input #1, tmpstr

        'trim the leading spaces
        tmpstr = LTrim$(tmpstr)

        ' prepare to loop numerous times through a line
        rflag = True
        Do While rflag = True

```



```

        ' yield control to Operating System
        If evtCounter Mod 100000 = 0 Then
            intEvts = DoEvents()
        End If
        rflag = False

        ' replace any double spaces with single spaces
        tmpresult = Replace(tmpstr, "  ", " ")
        tmpstr = tmpresult

        ' check to see if we have any remaining double spaces
        lngPos = InStr(tmpstr, "  ")
        If lngPos > 0 Then

            ' yes we have more double spaces, flag it
            rflag = True
        End If
    Loop

    ' we have exited the loop and only single spacing remains.
Write it out.
    Print #2, tmpstr
    lngRcd = lngRcd + 1

    If lngRcd Mod 1000 = 0 Then
        Form1.Frame1.Caption = "Processed " & CStr(lngRcd) & "
records."
    End If

    Loop

    ' close the files
    Close #1
    Close #2
    MsgBox "Finished clearing spaces from input file"
    Form1.MousePointer = 0

End Sub

Public Sub CommaDelimit()

    Dim tmpstr As String, tmpstr2 As String
    Dim tmpresult As String
    Dim hexswitch As Boolean
    Dim Outarray() As String
    Dim Tetcounter As Long
    Dim hexcounter As Long
    Dim outcounter As Long

    'common dialog control
    Form1.dlgCommonDialog1.InitDir = App.Path
    Form1.dlgCommonDialog1.Filter = "All Files (*.*)|*.*"
    Form1.dlgCommonDialog1.ShowOpen
    If Len(Form1.dlgCommonDialog1.FileName) = 0 Then
        Exit Sub
    End If

```

```

End If
sFile = Form1.dlgCommonDialog1.FileName

Open sFile For Input As #1
Form1.dlgCommonDialog1.Filter = "All Files (*.*)|*.*"
Form1.dlgCommonDialog1.ShowSave

vfile = Form1.dlgCommonDialog1.FileName
Open vfile For Output As #2

outcounter = 0
hexswitch = False
Form1.MousePointer = 11
Do While Not EOF(1)

    'This is the stuff we do for the normal parts of the code,
trimming etc
    If hexswitch = False Then      ' starthexswitch

        ' read our line
        Line Input #1, tmpstr
        'trim the leading spaces
        tmpstr = LTrim$(tmpstr)
        'change delimitation from space to comma
        tmpresult = Replace(tmpstr, " ", ",")

        'check to see if we found the marker for the hexahedral
nodes
        'if we do, flag our start.
        If tmpresult = "ELEMENTS/CELLS,2.1.6" Then
            hexswitch = True
        End If

    Else                            'else hexswitch

        'do this stuff if we are reading the special hex section

        'we dont know if this is a hex-section or a tetsection, or
even the end
        Line Input #1, tmpstr

        ' if its end of section we are done
        If tmpstr = "ENDOFSECTION" Then
            hexswitch = False
            tmpresult = tmpstr
        Else
            outcounter = outcounter + 1
            'well we are still in the nodes section at least

            'determine if we are looking at a hex/tet node

            'first we might as well delimit it
            'trim the leading spaces
            tmpstr = LTrim$(tmpstr)

            'change delimitation from space to comma

```

```

        tmpresult = Replace(tmpstr, " ", ",")

        'parse into an array so we can check
        Outarray = Split(tmpresult, ",")

        'its a zero based array, and the number of vertices is
the 3rd term
        If Outarray(2) = "8" Then
            hexcounter = hexcounter + 1
            ' we have an 8 vertex element. due to the way the
input files seem to be written
            ' we need to read one more line because it got
wrapped

            Input #1, tmpstr2
            tmpstr2 = Trim$(tmpstr2)

            'append the 8th vertex ID to the other 7
            tmpresult = tmpresult & "," & tmpstr2
        Else
            Tetcounter = Tetcounter + 1
        End If ' finished the 8-vertex node handling
    End If ' call for the end-of-section IF
End If

    'now re-write the whole thing out like we want
    Print #2, tmpresult

Loop
Form1.MousePointer = 0
Close (1)
Close (2)
MsgBox "Finished parsing "
Form1.Frame1.Caption = CStr(outcounter) & " elements written"
Form1.Label1.Caption = CStr(hexcounter) & " hex elements"
Form1.Label3.Caption = CStr(Tetcounter) & " tet elements"

End Sub

```

```

Public Sub ReadCommaDelimit()

```

```

    ' Ok hopefully the user has prepped the comma delimited file
    ' check that the output file was written

```

```

    Dim v1 As String
    Dim v2 As String
    Dim v3 As String
    Dim v4 As String
    Dim v5 As String
    Dim v6 As String
    Dim v7 As String
    Dim v8 As String

```

```

    If vfile = "" Then
        'common dialog control
    End If

```

```

        MsgBox "No input file was detected from a prep run. Please
select one."
        Form1.dlgCommonDialog1.InitDir = App.Path
        Form1.dlgCommonDialog1.Filter = "All Files (*.*)|*.*"
        Form1.dlgCommonDialog1.ShowOpen
        If Len(Form1.dlgCommonDialog1.FileName) = 0 Then
            Exit Sub
        End If
        vfile = Form1.dlgCommonDialog1.FileName
    End If

    ' open the file
    Open vfile For Input As #1

    ' read control info. We dont really need this stuff
    Dim CNTRLINFO1 As String
    Dim CNTRLINFO2 As String
    Dim CNTRLINFO3 As String
    Input #1, CNTRLINFO1, CNTRLINFO2, CNTRLINFO3

    ' read header info. We dont really need this stuff
    Dim HEDNUT1 As String
    Dim HEDNUT2 As String
    Dim HEDNUT3 As String
    Dim HEDNUT4 As String
    Input #1, HEDNUT1, HEDNUT2, HEDNUT3, HEDNUT4

    ' read the user title
    Dim UTITLE As String
    Input #1, UTITLE

    ' read data source and revision level again not really needed.
    Dim ASTRSK As String          ' some asterisks gambit neutral throws
up front
    Dim NPROG As String          ' this should read gambit if made by
gambit
    Dim FORMT As String          ' this should say neutral
    Dim REVISION As String       ' gambit version number
    Input #1, ASTRSK, NPROG, FORMT, REVISION

    ' read some date stuff.
    Dim strDate As String
    Dim strdate1 As String
    Dim strblank As String
    Input #1, strDate, strdate1, strblank

    Dim NUMNP As Long            ' total # of nodes
    Dim NELEM As Long            ' total # of elements
    Dim NGRPS As Long            ' total number of ele groups
    Dim NBSETS As Long           ' total number of bc sets
    Dim NDFCD As Long            ' total number of coord directions 2/3
    Dim NDFVL As Long            ' total number of velocity components 2/3

    ' read the headers
    Input #1, v1, v2, v3, v4, v5, v6

    ' now read their values

```

```

Input #1, v1, v2, v3, v4, v5, v6

NUMNP = CLng(v1)
NELEM = CLng(v2)
NGRPS = CLng(v3)
NBSETS = CLng(v4)
NDFCD = CLng(v5)
NDFVL = CLng(v6)

' read the end of section statement
Dim endsection As String
Input #1, endsection

' read the prep for coordinates. Not needed really.
Dim strNodal As String
Dim strCoor As String
Dim strVer As String
Input #1, strNodal, strCoor, strVer

' prepare to read coordinates
Dim NODEID As Long
' dimension the arrays dynamically
ReDim X(1 To NUMNP) As Double
ReDim Y(1 To NUMNP) As Double
ReDim Z(1 To NUMNP) As Double
Dim Iloop As Long, zloop As Long

'loop over each node
For Iloop = 1 To NUMNP
    Input #1, NODEID, v1, v2, v3
    X(Iloop) = CDBl(v1)
    Y(Iloop) = CDBl(v2)
    Z(Iloop) = CDBl(v3)
Next Iloop

' read the end of section statement
Input #1, endsection

' read elements/cells information
Dim ECELLS As String
Input #1, ECELLS, REVISION

' prep for reading vertices
ReDim VTX(1 To 8, 1 To NELEM) As Long
Dim jloop As Long
Dim kloop As Long

For Iloop = 1 To NELEM

    ' read global element number
    Input #1, v1
    NODEID = CLng(v1)

    ' read the element type and geometry type
    Dim NTYPE As Long
    Dim GTYPE As Long
    Input #1, v1

```

```

        NTYPE = CLng(v1)

        ' read the number of vertices
        Dim NVERTEX As Long
        Input #1, v1
        NVERTEX = CLng(v1)

        ' read the vertices

        If NVERTEX = 4 Then
            Input #1, v1
            VTX(1, NODEID) = CLng(v1)
            Input #1, v1
            VTX(2, NODEID) = CLng(v1)
            Input #1, v1
            VTX(3, NODEID) = CLng(v1)
            VTX(4, NODEID) = CLng(v1)
            Input #1, v1
            VTX(5, NODEID) = CLng(v1)
            VTX(6, NODEID) = CLng(v1)
            VTX(7, NODEID) = CLng(v1)
            VTX(8, NODEID) = CLng(v1)

        ElseIf NVERTEX = 8 Then
            For jloop = 1 To 8
                Input #1, v1
                VTX(jloop, NODEID) = CLng(v1)
            Next jloop
        Else
            MsgBox "Element type not handled"
            End
        End If

    Next Iloop

    ' read the end of section statement
    Input #1, endsection

    Dim t1 As String
    Dim t2 As String
    Dim t3 As String
    Dim t4 As Long
    ' this prepres us for boundary groups which is what we need for
    CONTAB
    ' begin by dynamically sizing contab and nodetype arrays
    ReDim NODETYPE(NUMNP) As Long
    ReDim CONTAB(1 To 6, 1 To NUMNP) As Long

    For zloop = 1 To NGRPS

        ' read in element group information
        ' write this out to a separate text file
        ' it will probably be useful to specify CHAD arrays within CHAD

        Input #1, t1, t2, t3
        Input #1, t1, t2, t3
        Input #1, t4, t2, t3
    
```

```

    Input #1, t1, t2
    Input #1, t2
    Input #1, t1

    vfile = t2 & ".dat"

    Open vfile For Output As #31

    For Iloop = 1 To t4
        Input #1, t1
        Print #31, t1
    Next Iloop
    Input #1, endsection
    Close #31

Next zloop

' preset nodetype array as interior nodes
' when model is z-symmetric change these nodes to free-slip
    For Iloop = 1 To NUMNP
        If (Form1.Zsymm.Value = 1) Then
            NODETYPE(Iloop) = 21
        Else
            NODETYPE(Iloop) = 0
        End If
    Next Iloop

' load contab with x-y-z freedom
    For Iloop = 1 To NUMNP
        CONTAB(1, Iloop) = 1
        CONTAB(4, Iloop) = 1
        If Form1.Zsymm.Value = 0 Then
            CONTAB(6, Iloop) = 1
        Else
            CONTAB(6, Iloop) = 0
        End If
    Next Iloop

Dim BCNAME As String
Dim ETYPE As Long
Dim NENTRY As Long
Dim NVALUES As Long
Dim BCCODE As Long
Dim tmpNType As Long
Dim tmpstr As String
Dim tmpstr2 As String
Dim NID As Long
Dim TMPNODE As Long
Dim version As String
Dim lngPos As Long
Dim wCounter As Long
Dim intCounter As Long
Dim oCounter As Long

For kloop = 1 To NBSETS
    Input #1, tmpstr, tmpstr2, version

```

```

Input #1, BCNAME, ETYPE, NENTRY, NVALUES, BCCODE

' determine what kind of BC set we are looking at
' flag what the node types will be
tmpstr = "inlet"      ' inflow
lngPos = InStr(BCNAME, tmpstr)
If lngPos > 0 Then
    NID = 41
End If
tmpstr = "wall"       ' freeslip
lngPos = InStr(BCNAME, tmpstr)
If lngPos > 0 Then
    NID = 31
End If
tmpstr = "outlet"     ' outflow
lngPos = InStr(BCNAME, tmpstr)
If lngPos > 0 Then
    NID = 51
End If
tmpstr = "symmet"     ' symmetry
lngPos = InStr(BCNAME, tmpstr)
If lngPos > 0 Then
    NID = 21
End If

vfile = BCNAME & ".dat"
Open vfile For Output As #33

For Iloop = 1 To NENTRY
    ' read the value
    Input #1, TMPNODE
    Print #33, TMPNODE

    NODETYPE(TMPNODE) = NID

    ' apply the contab
    ' if its wall we zero contab, othewise, leave it alone
    If NID = 21 Or NID = 31 Then
        CONTAB(1, TMPNODE) = 0
        CONTAB(4, TMPNODE) = 0
        CONTAB(6, TMPNODE) = 0
    End If
    Select Case NID
    Case 21
        wCounter = wCounter + 1
    Case 41
        intCounter = intCounter + 1
    Case 51
        oCounter = oCounter + 1
    End Select
Next Iloop

' read the end of section statement
Input #1, endsection
Close #33

```



```

Next kloop

' we exited the do loop
' there is nothing left to do
' so end
Close #1
Close #2

Call WriteCHAD(NUMNP, NELEM)

End Sub

Public Sub WriteCHAD(ByVal NNODES As Long, ByVal NELEMS As Long)

    Dim i As Long
    Dim multflag As Long
    Dim t_mult As Boolean
    Dim nstring As String

    'common dialog control
    Form1.dlgCommonDialog1.DialogTitle = "Write the name of the CHAD
output file"
    Form1.dlgCommonDialog1.InitDir = App.Path
    Form1.dlgCommonDialog1.Filter = "All Files (*.*)|*.*"
    Form1.dlgCommonDialog1.ShowSave
    If Len(Form1.dlgCommonDialog1.FileName) = 0 Then
        Exit Sub
    End If
    vfile = Form1.dlgCommonDialog1.FileName

    ' open the file
    Open vfile For Output As #1

    ' write the mesh dimensions
    Print #1, Spc(1); NELEMS; NELEMS; NNODES; NNODES; 0; 0; 0

    ' write the X data
    Print #1, "X"
    For i = 1 To NNODES
        Print #1, X(i)
    Next i

    ' write the Y data
    Print #1, "Y"
    For i = 1 To NNODES
        Print #1, Y(i)
    Next i

    ' write the Z data
    Print #1, "Z"
    For i = 1 To NNODES
        Print #1, Z(i)
    Next i

```

```

Next i

'write the node vertex info
'note the change from GAMBIT to CHAD in vertex order
Print #1, "NODESV"
For i = 1 To NELEMS
    Print #1, VTX(1, i); " "; VTX(2, i); " "; VTX(4, i); " ";
VTX(3, i); " "; VTX(5, i); " "; VTX(6, i); " "; VTX(8, i); " "; VTX(7,
i)
Next i

'write color info
Print #1, "ICOLORN"
nstring = CStr(NNODES) & "*1"
Print #1, nstring

'write material info
Print #1, "MATNON"
nstring = CStr(NNODES) & "*1"
Print #1, nstring

'write REGNON info
Print #1, "REGNON"
nstring = CStr(NNODES) & "*1"
Print #1, nstring

'write nodetype info
Print #1, "NODETYPE"

'write contab info
Print #1, "CONTAB"
For i = 1 To NNODES
    Print #1, CONTAB(1, i); " "; CONTAB(2, i); " "; CONTAB(3, i); "
"; CONTAB(4, i); " "; CONTAB(5, i); " "; CONTAB(6, i)
Next i

'write eps info
Print #1, "EPS"
nstring = CStr(NNODES) & "*0.0"
Print #1, nstring

'write material info
Print #1, "P"
nstring = CStr(NNODES) & "*155000000."
Print #1, nstring

'write rho info
Print #1, "RHO"
nstring = CStr(NNODES) & "*1000."
Print #1, nstring

'write T info
Print #1, "T"
nstring = CStr(NNODES) & "*300."
Print #1, nstring

'write tke info

```

```

Print #1, "TKE"
nstring = CStr(NNODES) & "*0.0"
Print #1, nstring

'write U info
Print #1, "U"
nstring = CStr(NNODES) & "*0.0"
Print #1, nstring

'write V info
Print #1, "V"
nstring = CStr(NNODES) & "*0.0"
Print #1, nstring

'write W info
Print #1, "W"
nstring = CStr(NNODES) & "*0.0"
Print #1, nstring

'write ICOLORE info
Print #1, "ICOLORE"
nstring = CStr(NNODES) & "*0.0"
Print #1, nstring

'write REGNOE info
Print #1, "REGNOE"
nstring = CStr(NNODES) & "*0.0"
Print #1, nstring

'end the input
Print #1, "end"
Close #1
MsgBox "Finished writing CHAD mesh file!"
'End
End Sub

```

Appendix C: CONTAB

C.1 Velocity Constraints

Velocity boundary conditions in CHAD are enforced by multiplying the computed nodal velocity, $\bar{\mathbf{u}}_c$, against a 3 x 3 symmetric constraint tensor array called CONTAB, N. CONTAB is essentially shorthand for CONstraint TABLE. Defining a unit vector as $\bar{\mathbf{n}}$,

$$\bar{\mathbf{u}} = \bar{\mathbf{u}}_c - (\bar{\mathbf{u}}_c \cdot \bar{\mathbf{n}}) \bar{\mathbf{n}} = (\mathbf{I} - \bar{\mathbf{n}}\bar{\mathbf{n}}) \cdot \bar{\mathbf{u}}_c = \mathbf{N} \cdot \bar{\mathbf{u}}_c \quad (\text{C.1})$$

the CONTAB array is given by

$$\mathbf{N} = \begin{bmatrix} 1-n_x^2 & -n_x n_y & -n_x n_z \\ -n_x n_y & 1-n_y^2 & -n_y n_z \\ -n_x n_z & -n_y n_z & 1-n_z^2 \end{bmatrix} = \text{CONTAB} \quad (\text{C.2})$$

Individual array entries exist for all nodes in the domain. Since the tensor is symmetric only six entries are kept. Therefore, the dimensions of CONTAB are 6 x nnodemax where nnodemax is the maximum number of nodes residing on a processor (CONTAB is a distributed array.) The values of CONTAB are given as follows

$$\begin{aligned} \text{CONTAB}(1, v) &= 1 - n_x^2 \\ \text{CONTAB}(2, v) &= -n_x n_y \\ \text{CONTAB}(3, v) &= -n_x n_z \\ \text{CONTAB}(4, v) &= 1 - n_y^2 \\ \text{CONTAB}(5, v) &= -n_y n_z \\ \text{CONTAB}(6, v) &= 1 - n_z^2 \end{aligned} \quad (\text{C.3})$$

were as usual v represents a node id.

The CONTAB array is read in as part of the mesh file. As such, it must be determined before CHAD is executed. When a node is no-slip, all entries of the CONTAB array are zero for that node. When a node is free-slip and does not lie at the intersection of multiple boundary condition types, $\bar{\mathbf{n}}$ is normal to the boundary. When a free-slip node lies at the intersection of two boundary nodes $\bar{\mathbf{n}}$ cannot be normal to multiple boundary surfaces at the same time.

A method for internally computing the CONTAB array was developed at Argonne National Lab that resolves the issue of specifying CONTABs at conflicting boundary conditions (Tzanos-c, 1997). In this method, cell component areas are defined by setting

$$\begin{aligned} a_x &= \sum_i \mathbf{v}_{ix} A_i \\ a_y &= \sum_i \mathbf{v}_{iy} A_i \\ a_z &= \sum_i \mathbf{v}_{iz} A_i \end{aligned} \tag{C.4}$$

where A_i is the i th boundary surface element of the node being considered, and \mathbf{v}_{ix} , \mathbf{v}_{iy} , and \mathbf{v}_{iz} are the components of the unit vector normal to the surface A_i . Then, the components of the normal vector may be defined as

$$\begin{aligned} n_x &= \frac{a_x}{\left(a_x^2 + a_y^2 + a_z^2\right)} \\ n_y &= \frac{a_y}{\left(a_x^2 + a_y^2 + a_z^2\right)} \\ n_z &= \frac{a_z}{\left(a_x^2 + a_y^2 + a_z^2\right)} \end{aligned} \tag{C.5}$$

Appendix D

The ACG Group CHAD User's Guide

Appendix E: Installation and Configuration of Systems to Run CHAD	238
E.1 Software Overview	238
E.2 Software Installation Tips	241
E.3 Running CHAD Sample Problems	251
E.4 Creating Models in CHAD	251
E.5 The CHAD Input File	260
E.6 Boundary Conditions in CHAD	2699
E.7 Running CHAD	276
E.8 Post-Processing CHAD	278
E.9 Internals of CHAD	280

Preface

Installing, compiling, testing, using, and maintaining a large commercial-type code such as CHAD and PETSc on either a workstation or a parallel computing cluster is a time-consuming, complicated and often times frustrating task. Intricacies of hardware-software interactions on ever-changing system configurations with continual software releases require in depth knowledge typically found more in computer science majors than nuclear engineers. In the experience of this author, many computer science majors have found themselves inexperienced in matters discussed within this manual.

Once the hurdles of installing and configuring CHAD and its supporting software have been conquered, the reader will find themselves ready to tackle CHAD itself, a no less daunting task with the limited documentation and support available. Running sample problems provided with the CHAD distribution is certainly easy enough. Modifying the problems is slightly more complicated a task, with no manual in which to look up input flags or parameters. Creating new problems is an even more time consuming task. With only extremely rudimentary mesh generation capabilities provided with CHAD, most meshes must be developed using other software and imported into CHAD. Applying complex boundary conditions or sources is yet another, deeper step into understanding CHAD, which requires explicitly modifying CHAD's source code. While this provides flexibility, it also requires a thorough understanding of CHAD, knowledge which may only be obtained through extensive reading of the source code and some extrapolation from a draft version of CHAD's physics manual.

When a new CHAD user succeeds in solving a model in CHAD, viewing CHAD's results and having confidence in the solution is the next part of the learning process. While installing and using software required to visualize CHAD simulation results is among the easiest tasks a new CHAD user will face, extracting a user's own information and visualizing that information is something that can prove to be elusive unless a user knows exactly what they are looking for.

This user's guide is provided to assist the novice CHAD in becoming familiar with the code, its administration, and its use. In particular, this document hopes to significantly reduce the amount of time required for a new CHAD user to configure and install CHAD and its supporting software. Additionally, through this manual, the reader should quickly be able to effectively and efficiently run CHAD for a variety of problems, including new problems defined by the new CHAD user.

What this document does not intend to do is duplicate information already available about CHAD, such as the numerical methods, programming styles, and physics equations. This information may be found in the readily available documentation and literature.

Appendix E: Installation and Configuration of Systems to Run CHAD

E.1 Software Overview

A plethora of software is required should CHAD be installed on a new system with a typical suite of features. C, FORTRAN 77 and FORTRAN 90 compilers are all required, along with linear algebra packages and message passing libraries. By themselves, these software packages are generally fully functional, well developed pieces of software. Getting them to all work together typically requires many hours of work. We briefly review each of the independent pieces of software providing some useful comments and firsthand experience with them. Later in this document technical issues that accompany advanced compilation options and linking of the software are more thoroughly discussed.

E.1.1 Operating System

All the work described within this document has been performed on Red Hat Linux versions 7.2 and 7.3. CHAD was also compiled on Red Hat Linux 8.0, though no work was actually performed on that system. At this time, it is thought that this is the only work done with CHAD on Red Hat 7.2 or 7.3. Other operating systems have been minimally used, such as Solaris 2.8, however they are not discussed within this document. CHAD was designed to be portable across computing environments, so portability should not be a significant problem.

No particular difficulties were encountered using either the Linux or Solaris as operating systems for CHAD. Of note, however, is that Red Hat no longer provides technical support for these versions of Linux and as such new installations of CHAD are unlikely to be done on this operating system.

E1.1.2 FORTRAN Compilers

CHAD is a large and modern FORTRAN 90 code. The work performed here used Absoft's Pro FORTRAN compilers versions 7.5 and 8.0, depending on the computer system discussed. The University of Illinois' Advanced Computation Group uses version 7.5 while the University's Computational Science and Engineering's Turing computing cluster uses 8.0. Both compilers appear to work fine with CHAD. However, Absoft's version 7.5 compilers contain a rather large bug in the compiler's front end which necessitates patching should they be used.

E1.1.3 C Compilers

CHAD is written using FORTRAN files that require C preprocessing. Various GNU C compilers between versions 2.7.X and 3.3.3 have successfully be used to that end. While the compilers performed satisfactorily, some of the coding practices used in CHAD 5.0 require tedious modifications to work with version 3.3.3 of the C compiler, a topic discussed in further detail later in this manual. Additionally, some of the code in the PGSLibrary, of which CHAD is dependent on, requires C compilation.

E1.1.4 CHAD

CHAD is the basis code for this software. It is an export controlled public domain software available from Los Alamos National Lab. It requires a C preprocessor, FORTRAN 90/95, and version of MPI if it is to be run in parallel. Optionally, the PETSc software package may be used along with the BLAS and LAPACK linear algebra libraries.

E1.1.5 PGSLib

PGSLib, the Parallel Gather/Scatter Library, is a communication library used to improve the parallel MPI performance on clusters. CHAD requires either PGSLib or CommLib to perform a multitude of parallel communication tasks. Contacts within Los Alamos by those familiar with CHAD were unable to provide a location from which CommLib is obtainable, therefore we used PGSLib to us to fulfill this role.

PGSLibrary can compile both parallel and serial libraries that may be linked with CHAD. If CHAD is being run in serial mode, the serial PGSLibrary should be linked. If it is being run in parallel, PGSLibrary requires the linking of an MPI implementation. Installation of PGSLib requires C, a C preprocessor, FORTRAN 77, and FORTRAN 90.

E1.1.6 MPI & MPICH

PGSLibrary, and optionally PETSc, require an implementation of MPI. All our experience has been with a version of MPI called MPICH that may be obtained on the internet at Argonne National Laboratory's web page. There have been no issues

concerning different versions of MPI. Three versions of MPICH have been independently used, within the ACG, at ANL, and on the CSE Turing cluster.

E1.1.7 PETSc

If the matrix solver implementation is intended to be used with CHAD, PETSc will need to be installed on your system. PETSc requires, in addition to C and FORTRAN compilers, an implementation of MPI, BLAS, and LAPACK. BLAS and LAPACK are freely available on the web.

E1.1.8 BLAS & LAPACK

The Basic Linear Algebra Set and Linear Algebra Package are highly efficient FORTRAN 77 libraries available freely on the web. They are required by PETSc.

E.2 Software Installation Tips

Each piece of software described earlier comes with its own installation guide. In most cases, these guides direct you through a semi-automated process that requires little user interaction. However, in our experience, and especially with our experience installing CHAD, several things tend not to be mentioned or emphasized sufficiently. These little issues can turn out to be exceptionally annoying and frustrating details that can leave one working for countless hours.

E.2.1 Before You Begin

If you are required to install any of the above software packages, ensure you follow several important guidelines:

1. Patch your operating system
2. Patch your compilers
3. Beware of previously installed software.

Our experience with installing CHAD and other software involves a number of technical difficulties, of which quite a few were solved by simply patching software. If you are installing software on other a system maintained by someone else, check to make sure they are running patched versions of software. We have encountered software at ANL and on the CSE Turing cluster that contained issues preventing us from installing CHAD that was easily resolved by patching compilers.

E.2.2 Uniformity in Software Installations

CHAD requires a number of supporting software packages from PGSLibrary to MPI to BLAS/LAPACK. It is absolutely critical that when installing software packages, that they be installed using the same compiler options. Software packages often will not link together if they were not compiled using the same compiler flags. In many instances, it is not readily apparent which options were used to create object and library files, however, the linking of object files requires consistency between all of the software in order to create a working executable. In particular, it is important to understand how your C compiled and FORTRAN compiled object files will interact when linking is attempted.

For example, PGSLibrary is a combination of F77, F90 and C-language code. For successful linking between object files created by different compilers, the names of the routines within the object files (known as symbols) must be identical. The difficulty in linking the symbols between object files results from a compiler's renaming of the symbol name to some other name. For example, C compilers tend to append an underscore “_” to the front of a symbol. Similarly, most FORTRAN compilers convert symbol names to upper-case characters. This process of renaming a symbol is called “name mangling” or “decorating”. There are no rules or standards for mangling of symbol names, and compilers vary in how they do it. While some breeds of compilers (C and pre-pended underscores, FORTRAN and case conversion etc.) may generally follow certain trends, no one rule covers them all. As a consequence, the linking of mixed language libraries is left to the user to resolve.

CHAD requires linking to PGSLibrary, a mixed language library package, which in turn depends upon MPICH. Additionally, CHAD may depend on PETSc, which separately depends upon MPICH and BLAS/LAPACK. The linking of all these independently compiled libraries together fast becomes a complicated task.

At very minimum expect to be forced to link F77, F90, and C objects all from within PGSLib to both an external legacy support library provided by Absoft as well as MPICH. Linking these together in a fashion that allows one to run PGSLib should be sufficient to link to CHAD successfully.

If at all possible, independently compile all your software before you begin so that you are sure your code will be compiled using the same parameters.

For the most part, installing the aforementioned pieces of software was easy and without trouble. However, there are a few things one should be aware of:

- ◆ Use the most recent versions of your compilers and tools to ensure they are ‘bug free’ and will not later be phased out or deprecated.
- ◆ Be aware of your environment variables and `PATH`. On several occasions, improper path statements caused incorrect compilers to be used.
- ◆ Be sure your compilers are the ones you think you are using by performing a `which` `gcc` or similar command.

E.2.3 Compilation Issues

E.2.3.1 Compiling CHAD

The CHAD distribution package includes a makefile utility, called *makemakefile*, which generates the makefile that is then used to compile CHAD. This makefile utility prompts the user with selected questions regarding their operating environment and desired features of CHAD, and then attempts to configure a makefile script. If you are lucky, the *makemakefile* utility will create your makefile which you may then use to compile CHAD. This utility contained errors on the Solaris operating system, but runs without error on the Linux platforms.


```
file      #|i|n|c|l|ü|d|e|      |"m|a|c|r|o|s|.|H|H|"_ü|s|e|      |
|K|i|n|d|s|_|m|o|d|u|l|e|_|c|  !wrong way
1|2|3|4|5|6|7|
```

```
_|_|_|_|_|_|_|u|s|e| | | K|i|n|d|s|_|m|o|d|u|l|e|_|c !correct  
way  
  
1|2|3|4|5|6|7|.
```

Newer versions of the gcc compiler do not accept multiline string literals. Thus lines such as the following

245

```

& /, a
& )"
& ) blank_line

```

wind up appearing as

```

write(Inout_String,
& "(/, ' System Totals:', "
& /, a
& )" "
& ) blank_line

```

after preprocessing. The solution to this is to either rewrite the strings or manually fix the strings in the resulting .F files. CHAD contains quite a bit of programming in the first form above. Rewriting all of CHAD to conform to the requirements of gcc would take a very long time and it is not recommended that one spend their time in doing so. However, if one frequently makes changes to subroutines that will be recompiled, it makes sense to rewrite those blocks of code as after every compilation the superfluous quotation marks would need to be removed before the FORTRAN compiler pass.

Compiler Flags

CHAD has been compiled with a variety of compiler flags. Their use is listed below.

- ◆ -c compile object files only, saving linking for later
- ◆ -g generate debugger information for the xfx debugger

- ◆ -V output compiler version number
- ◆ -m(1-4) Dictates the level of compiler messages provided. 0 being the most and 4 being the least information.
- ◆ -O(1-2) optimize the code. Optimization will not function in Absoft Pro FORTRAN 7.5 without a patch.

Several compiler options did not work with CHAD, in particular array bound checks prevented CHAD from compiling successfully.

E.2.3.2 PGSLibrary

PGSLibrary is the likely the most difficult piece of software one will need to compile. One reason is that PGSLib does not provide a compilation script for Absoft compilers on a Linux operating system. Therefore, it is up to the user to configure the makefiles and build scripts for this type of system so that the correct name mangling occurs. A second reason is PGSLib is a mix of C-language and FORTRAN code that requires naming conventions between function names to match throughout the code. This often has to be accomplished either through a rewrite of the code, or via compiler flags. Thirdly, there is very little in the way of an installation guide, so any difficulties are up to the user to resolve. Finally, the library comes with little to no customer support.

Despite the difficulties with PGSLibrary a successful compilation can be obtained. A few tips are suggested below to help ensure you are on the right track.

PGSLibrary requires a mix of C and FORTRAN code to work in sync. The name mangling of the resulting library symbol names should match both internally and against the installation of MPI or MPICH. The library names can be obtained using the `nm` utility. It is useful to route the output of `nm` to a file for review when difficulties are encountered. For example,

```
nm libpgslib-par.a > nm.out
```

outputs the internal names of routines in `libpgslib-par.a` into a file called `nm.out`. Inspection of `nm.out` allows the user to visualize how the compilers are writing the routine names. Within the library, the names should of course match against themselves. However, it is entirely possible that the library may be built but that the internal references do not match. This error will appear when one attempts to run the sample problems provided with PGSLib. A few things to check include

- ♦ Check leading and trailing underscores within the compiled library and against any MPI library calls. You will likely need to check the MPI libraries with the `nm` utility. Absoft provides compiler flags to either append or prepend underscores, as well as allowing the user to manually specify their own prefix or suffix. Compiler flags include `-YEXT_SFX='_'` and `-B108`.
- ♦ Check the case of external symbols. They may need to be either upper or lower case. The appropriate compiler flag is `-YEXT_NAMES="UCS|LCS|ASIS"` for upper case, lower case, and "as is" for no change.

- ◆ It is important to remember that the C-language leaves names “as is” so one can manually change the code to reflect this behavior if the name mangling scheme is overly complicated. The best way to resolve this is to have all applications compiled before you begin with the same compiler sets and the goal of harmonious interaction. This author has successfully compiled PGSLibrary on two separate parallel computing systems by internally matching symbols within PGSLibrary and against MPICH. This has generally been accomplishing by changing the case and underscores of C routines as required. Most of the C routines are of mixed case and required conversion to either upper or lower case. Additionally, a few C routines required the application of an underscore to their name. In some places this proves confusing as routine names are actually constructed during the compilation phase and are not hardwired within the code. Attempting to fix the resulting routines is tiresome at best, and if PGSLib needs recompiled this fix needs repeated. It is much better to modify the source that is constructed during the preprocessing stage. This particular user spent was not fortunate enough to have noticed that the routines were built during preprocessing, and therefore spent needless hours rebuilding the installation.

- ◆ PGSLibrary uses the `ETIME` function. It is not an intrinsic function in the Absoft compilers. However, Absoft does provide support for `ETIME` via a support library, `libU77`, which requires linking to the PGSLib routines. The `ETIME` function in that library comes in three forms: `'ETIME'`, `'ETIME_'`, and

`'etime_'`. This makes it clear that when the PGSLibrary is linked externally against either your application or MPI the choice of compiler flags must not be both lower case and no trailing underscore because there is no reference to `'etime'`. `ETIME` issues can be resolved by renaming the function within PGSLibrary or by choosing another naming convention.

- ◆ Manually run the parallel test examples in addition to using the batch script provided. On the UIUC's CSE Turing cluster, it was found that the examples will claim to run correctly, but in reality they will fail during the `MPI_Init` phase.
- ◆ The order of linking external libraries is important with the Absoft compiler. Symbols are checked against libraries that *trail* it in the linking order. If no reference is found an error will be produced. If the library is already included before the unresolved reference, it may need to be included again after any unresolved reference.

E.2.3.3 MPICH, PETSc and BLAS/LAPACK

The remaining programs, MPI, PETSc and BLAS/LAPACK all compile fairly easily and require few special considerations. Nevertheless there are a few suggestions to keep one on the right track

Be sure to use `mpif77` and `mpif90` for your MPI compilations. They ensure you are using the correct compiler and link in any appropriate support libraries.

The PETSc customer support is excellent. Do not hesitate to email them with any questions you may have. You will often receive a reply within 30 minutes.

E.3 Running CHAD Sample Problems

CHAD distributions come with 11 sample problems, of both density-based and pressure-based schemes. Only the pressure-based version of CHAD has been made available to the ACG Group.

After successfully completing a compilation of CHAD, the sample problems should be run and the results reviewed. Sample results may be checked against “blessed results” using the `User_test` routines. Caution should be exercised using the testing routines however, and it is encouraged to independently verify results against published benchmark results. Most all of the test problems are not applicable for nuclear thermal-hydraulics and therefore an independent set of benchmark cases is advised to be used before any actual work begins.

E.4 Creating Models in CHAD

Building CHAD models is a several step process. First, geometry for the model must be created. Second, this geometry must be meshed. Thirdly, this mesh must be

output into a format suitable for CHAD to read and use. Finally, an input file must be created that describes the problem. Each of these areas deserves its own attention.

For the purposes of this research, the commercial code Gambit, a preprocessor to Fluent, was used to generate geometries and mesh them. A translator developed in-house was used to convert a form of the Gambit output file into a CHAD readable form. See Appendix B for further details regarding this translator. Without using a mesh generation tool, such as Cubit or the Gambit + mesh translation, it is almost impossible to construct useful test cases.

E.4.1 Creating Model Geometries

The first step in building a CHAD model is to create the geometry that requires meshing. This is almost always done using a computer aided design program (CAD). The meshing industry exists semi autonomously from the CAD industry so a variety of geometry building programs, such as ProEngineer, are available that can export models into a format suitable for a meshing program. The choice of your CAD program depends in part on your choice of your meshing program. Some meshing programs, such as ICEM-CFD's recently have been packaged with a basic CAD program built in. This will simplify model building for simple geometries by removing unnecessary features not required for CFD models, and provide relatively seamless flow into the next step of meshing the geometry.

It is a good idea to plan your geometry on paper before you begin creating it in a CAD program, so that time consuming mistakes may be avoided during geometry creation.

E.4.2 Meshing Geometries

Once a model geometry has been created, it needs meshed. Mesh programs are a field all their own, and selection of a meshing program is another decision a user must make. However, in the end, the mesh requires translation into a format readable by CHAD and at this time it appears only ICEM-CFD provides a commercial translator that will generate meshes for CHAD. Of course, any other meshing program may be used but it would then be up to the CHAD user to manually write a translator to convert a generic mesh format into a format suitable for CHAD.

A simple FORTRAN meshing tool is provided by CHAD to generate basic blocks, and a basic mesh generation program was also written at ANL to do the same. Both of these tools may be used to construct simple cavity or duct flow types of problems. This of course requires a slightly more in depth understanding of boundary conditions and the delving into more coding, whereas modern meshing tools are likely to have graphical user interfaces. At least one advantage of a graphical interface is the ability to visualize your mesh thereby reducing the number of accidental mesh errors. On more than one occasion an incorrect boundary condition specification has resulted from a typographical error in the custom mesh tools.

It is generally a good idea to begin with a coarse mesh and run them for short periods of time within CHAD to identify any problems with the mesh, and identify any areas that require further mesh refinement. This strategy will save time in the long run.

E.4.2.1 CHAD Mesh Files

A typical CHAD mesh file contains three sets of information. First, it contains grid information such as coordinates for vertices and cell information. Secondly, it contains initial values for variables within the code. Thirdly, it contains information about the nodal and elemental distribution of the domain in a parallel simulation.

The name of a CHAD mesh file is acquired through a parameter in the CHAD input file. After the CHAD input file is read, the subroutine `read_mesh` directs CHAD to open the mesh file and begin reading variables. The reading of this mesh file is a bit complex so if any particular details are sought about reading the mesh, this subroutine should be consulted.

The first line of the mesh file can be in either of two forms. The first form is the type found in the sample problems included with CHAD and contains several dummy variables. The format for this is

```
nelemax nelemax nnodemax nnodemax nparmax nconns nsp  
nsparen
```

where the first two variables are the number of elements in the problem, the second two are the number of nodes in the problem, `nparmax` represents a parent marker for

material interfaces (of which typical ACG users will not be using) `nconns` is the number of connections in the input file (which is generally also zero as CHAD will calculate it based on mesh data, and `nsp` is the number of species which will default to the number of species detected from the input file if set to zero. The final variable, `nsparen` is the number of user defined spare-nodal arrays. These arrays are used to store user calculated values. The default number of spare nodal arrays is three and the largest of spare nodal arrays values is always chosen (meaning the minimum number will always be at least three).

The next set of commands executed during the reading of the input file involves reading scalar arrays. CHAD will scan the input file for scalar array names, and upon finding them read them into the proper arrays. The scalar arrays typically read are nodal based arrays and are summarized in Table E.4.1

The arrays in the CHAD mesh file are marked by their name existing alone on a line, followed by the data to be placed within that array. The data to be loaded within that array may be formatted in two separate ways. The simplest is to explicitly list the value of each variable. The other way is to use the asterisk (*) to denote multiples of certain values. For example, the following two examples are equivalent:

`0.1, 0.1, 0.1, 0.1, 0.1` and

`5*0.1`

Table E.4.1 Common variables read from a CHAD mesh file

Variable	Size	Meaning
X	Nnodemax	X-coordinate of node
Y	Nnodemax	Y-coordinate of node
Z	Nnodemax	Z-coordinate of node
NODESV	8 x nelemax	list of nodes associated with element vertices [*]
ICOLORN	Nnodemax	node color after domain decomposition
MATNON	Nnodemax	node material number
REGNON	Nnodemax	node region number
NODETYPE	Nnodemax	node type of element
CONTAB	6 x nnodemax	symmetric constraint tensor
EPS	Nnodemax	turbulent kinetic energy dissipation rate
TKE	Nnodemax	turbulent kinetic energy
U	Nnodemax	U-velocity
V	Nnodemax	V-velocity
W	Nnodemax	W-velocity
P	Nnodemax	pressure
T	Nnodemax	temperature
RHO	Nnodemax	density
ICOLORE	Nelemax	element color after domain decomposition
REGNOE	Nelemax	element region number

^{*} See Appendix B for numbering scheme for elements of order lower than hexahedral.

The CONTAB array is a special, two-dimensional array used to apply velocity boundary conditions. It is a symmetric constraint tensor that contains six entries per node. Nodal velocities have constraints applied to them by dotting the CONTAB array into the component of velocity

$$\bar{\mathbf{u}} = \bar{\mathbf{u}}_c - (\bar{\mathbf{u}}_c \cdot \bar{\mathbf{n}}) \bar{\mathbf{n}} = (\mathbf{I} - \bar{\mathbf{n}}\bar{\mathbf{n}}) \cdot \bar{\mathbf{u}}_c = \mathbf{N} \cdot \bar{\mathbf{u}}_c$$

where

$$N = \begin{vmatrix} 1-n_x^2 & -n_x n_y & -n_x n_z \\ -n_x n_y & 1-n_y^2 & -n_y n_z \\ -n_x n_z & -n_y n_z & 1-n_z^2 \end{vmatrix} = \text{CONTAB}.$$

When a node is no-slip, CONTAB is zero. However, when a node is free-slip, CONTAB may defined by computing the components of a normal vector as

$$n_x = \frac{a_x}{(a_x^2 + a_y^2 + a_z^2)}$$

$$n_y = \frac{a_y}{(a_x^2 + a_y^2 + a_z^2)}$$

$$n_z = \frac{a_z}{(a_x^2 + a_y^2 + a_z^2)}$$

where a is the sum of the component face areas of a node given by

$$a_x = \sum_i \mathbf{v}_{ix} A_i$$

$$a_y = \sum_i \mathbf{v}_{iy} A_i .$$

$$a_z = \sum_i \mathbf{v}_{iz} A_i$$

Here, \mathbf{v}_{ic} are the components of the unit vector normal to the surface A_i . The CONTAB array may then be defined as

$$\begin{aligned} \text{CONTAB}(1, v) &= 1 - n_x^2 \\ \text{CONTAB}(2, v) &= -n_x n_y \\ \text{CONTAB}(3, v) &= -n_x n_z \\ \text{CONTAB}(4, v) &= 1 - n_y^2 \\ \text{CONTAB}(5, v) &= -n_y n_z \\ \text{CONTAB}(6, v) &= 1 - n_z^2 \end{aligned}$$

CHAD has various nodetypes which are defined in `nodetypes_module_c` and are used to specify node type entries in the `NODETYPE` section of the mesh file. Additional nodetypes may be added in this module. They are summarized below.

Table E.4.2: Nodetypes used in CHAD

Nodetype	Value	Meaning
<code>ntp_missing</code>	-1	missing (not in the problem)
<code>ntp_int</code>	0	interior node
<code>ntp_free</code>	11	free boundary node
<code>ntp_freeslip</code>	21	free-slip boundary node
<code>ntp_noslip</code>	31	no-slip boundary node
<code>ntp_inflow</code>	41	inflow boundary node
<code>ntp_outflow</code>	51	outflow boundary node
<code>ntp_uvwsrc</code>	61	velocity specified nodes
<code>ntp_tmprsrc</code>	71	temperature specified nodes
<code>ntp_mfracsrc</code>	81	species mass fraction specified nodes
<code>ntp_piston</code>	91	piston nodes
<code>ntp_kesrc</code>	101	k or epsilon specified nodes
<code>ntp_free_src</code>	111	energy source node for the Dante code
<code>ntp_freeslip_src</code>	121	energy source node for the Dante code
<code>ntp_intrfc</code>	1001	material interface node

* See Appendix B for numbering scheme for elements of order lower

E.4.2.2 Partitioning the Mesh

For parallel simulations, the distribution of nodes and connections across processors can greatly affect the overall time required to complete a simulation. A poor distribution results in excessive communication between processors. An ideal distribution minimizes this overall communication cost.

To achieve optimal distributions, generalized third party software programs have been developed which partition a mesh into a given number of domains. These domains present a minimal surface area between adjacent domains minimizing communication costs, while balancing the overall number of nodes between processors. One such program, METIS (Karypis and Kumar, 1998), has been used with CHAD to partition grids used in this dissertation.

METIS requires a rather simple input file requiring only a header line and an element vertex list. Based upon this input list, and several parameters provided at execution time, METIS creates a list where the index of the list indicates a nodes id number, and the value indicates the processor id that a node should end up on. The id of a processor is typically referred to as its ‘color’. This list should then be read into the `ICOLORN` section of the mesh file being used.

METIS may be linked at compile time to CHAD and the partitioning of the grid may be invoked at run time, though typically partitioning of a grid is done before this step. In this experience of this author, it was found to be more convenient to generate the METIS input file as part of the mesh translation step performed using the Gambit to CHAD mesh translation tool developed within this dissertation.

When CHAD reads in the nodal color array, `ICOLORN`, from the mesh file, it internally reorganizes the node numbering scheme used within the code using the

FORTRAN `SHUFFLE` intrinsic. In most cases, this reshuffling of the data does not affect simulation results. However, in the case external data files are used to flag certain nodes, for example, as boundary conditions, these arrays must be provided in the mesh file as opposed to read in later during code execution. The following example illustrates why.

Imagine a mesh has been developed using GAMBIT that contains two different boundary conditions. The mesh translation tool outputs the nodes that are part of these boundary conditions into files lists as lists. During code execution, these lists are read in during a pass through the input routine `User_in`, which modifies mesh data. In this routine, these nodes are collocated onto the IO processor and then distributed to their respective processors in a parallel simulation. When the mesh is not partitioned, this practice works as the nodes are not shuffled. However, when partitioning is done on the mesh, the partitioning occurs before the lists are read in `User_in`, and the resulting collocation and distribution occurs on an unshuffled list. Therefore, these lists should be provided as part of the mesh file, and stored in either the `SPAREN` arrays or `REGNON` arrays. If they are read in through the mesh file, they will reshuffled along with the rest of the mesh data.

E.5 The CHAD Input File

There is no publicly available user manual for CHAD, so there is no guide for creating an input file to specify model characteristics file data, or any other information used to control or set the problem. Furthermore, there is no graphical user interface to CHAD, so all input files must be constructed through a text editor. Nevertheless, some

parameters in an input file are known to be required, because they serve as boundary conditions and close the equations in the problem. For example, in a pipe flow, either the velocity or mass flow rate must be provided at the inlet. However, even though these it is known that these variables must be provided, the keywords used to specify them are actually unknown. Thus, with no reference point at which to begin, creating an input file by sifting through heaps of source code looking for the definition of a variable that matches your requirements is rather time consuming.

Several methods are available for creating an input file. One method is to modify an existing input file, and change it to match some new problem being defined. A second method of learning is by actually running a any CHAD sample problem. In doing so, CHAD echoes all of the parameters that could have been specified along with their current settings. This provides all the variable names that could have been specified and therefore provides a list. With this list, the user may then search the source code of CHAD to learn the meaning of the variable within the documentation. In some cases, even this is not sufficient. For example, one may know that the number of cycles of the code is available to be used as an iteration control, but it is unknown which data type is used to specify this. All possible input variables have therefore been tabulated in Table E.5.1, along with their data types, short definitions, and their default values.

Most, but not all variable descriptions may be found in the header file `DEFINE.HH`, as well as a very basic overview on how to generate an input file. However, this is not always sufficient as rarely are more than one or two sentences are

used to describe a variable. Therefore, its usually a good idea to `grep` a variable within the code and output all instances found, and then peruse the code where the variable is found and find out exactly how it is being used. This will be of huge assistance. In fact, it has proved so useful that it is actually a good idea to make a listing of every variable you ever look up. The user will find they frequently are investigating the usage of variables and saving `grep` results to `grep` files provides a quick and easy way of locating information.

Table E.5.1: Possible input variables to a CHAD simulation

Variable	Type	Default	Meaning
adb_wall	log	F	flag to make walls adiabatic
ANC4		0.00E+00	4 th order node coupling coeff.
AREA_PROJECTION	log	T	flag for computing projected median mesh area
ATDC	real	-1.80E+02	crank angle at inner dead center at zero time
CA_DUMP	real	7.20E+02	crank angle interval for dump files
CA_DUMP_START	real	-1.80E+02	crank angle after which dumps will start
CA_POST	real	7.20E+02	crank angle interval for post files
CA_POST_START	real	-1.80E+02	crank angle after which post processing will start
CAMAX	real	3.60E+02	max crank angle for problem termination
CARTL	real	0.00E+00	linear artificial viscosity coefficient multiplier
CARTQ	real	0.00E+00	quadratic artificial viscosity coefficient multiplier
CC_CONT_GLO	real	1.00E-04	cont. eqn. global convergence criterion
CC_CONT_LCL	real	1.00E-05	cont. eqn. local convergence criterion
CC_KE_GLO	real	1.00E-03	ke global convergence criterion

Table E.5.1 Continued

CC_KE_LCL	real	1.00E-04	ke local convergence criterion
CC_SPC	real	1.00E-03	speciens convergence criterion
CC_STRS_GLO	real	1.00E-03	stress-dev. global convergence criterion
CC_STRS_LCL	real	1.00E-04	stress-dev. Local convergence criterion
CC_TMP_GLO	real	1.00E-03	temp. global convergence criterion
CC_TMP_LCL	real	1.00E-04	temp. local convergence criterion
CC_UVW_GLO	real	1.00E-03	vel. global convergence criterion
CC_UVW_LCL	real	1.00E-04	vel. local convergence criterion
CHEMISTRY	char	uninitializ ed	type of chemistry
CINT	real	0.00E+00	anti-diff. mutiplier for material transport eqn.
CLENGTH	real	1.00E+00	length conversion factor
CMASS	real	1.00E+00	mass conversion factor
CONROD	real	1.70E-01	connecting rod length
CPU_DUMP	real	1.44E+04	cpu time intervals for dump files
CPU_MAX	real	hugenum	max cpu time
CTEMP	real	1.00E+00	temp conversion factor
CTIME	real	1.00E+00	time conversion factor
DELTALMAX	real	hugenum	max effective zone length for artificial viscosity
DELTALMIN	real	tinynum	min effective zone length for artificial viscosity
DT_DUMP	real	hugenum	dump time interval
DT_DUMP_STA RT	real	0.00E+00	time after which dt_dump starts
DT_POST	real	hugenum	post time interval
DT_POST_ACC URACY	real	hugenum	time accuracy factor for post processing
DT_POST_STA RT	real	-hugenum	time after which dt_post starts

Table E.5.1 Continued

DTCYC0	real	dtmax_def ault	time step size for cycle 0
DTMAX		hugenum	maximum time
DTMIN	real	1.0e- 4*dtcyc0	minimum time
DTRECMIN	real	tinynum	min time step based on recommended time
DTRST	real	-1.00E+00	time step size after restart
DUMPFIL	char	chad.dmp	name of dump file
EPSIN	real	0.00E+00	inflow boundary eps
EPSOUT	real	0.00E+00	outflow boundary eps
EPSSRC	real	0.00E+00	eps source
FLOWTYPE	char	laminar	type of flow
FTLAG	real	0.00E+00	tilde velocity factor for near-Lagrangian modes
GEOMETRY	char	uninitializ ed	indicates problem geometry
GMV_FORMAT	fmt	(1p,10e13. 5)	output format for post files
GRAD_LIMITER	char	MUSCL	gradient limiter options
HYDROTYPE	char	implicit	indicates type of hydrodynamics
Ictz	int	0	flag for selecting CTZ tilde scheme
imon_node_e	int	0	node ID for monitoring eps
imon_node_k	int	0	node ID for monitoring TKE
imon_node_p	int	0	node ID for monitoring pressure
imon_node_t	int	0	node ID for monitoring temperature
imon_node_u	int	0	node ID for monitoring u-velocity
imon_node_v	int	0	node ID for monitoring v-velocity
imon_node_w	int	0	node ID for monitoring w-velocity

Table E.5.1 Continued

IN_TYPE	int	1	indicates inflow boundary type
INTRFC_OPT	char	pequil_teq uil_vnosli	indicates options for interface exchange rates
INTRFC_RECONSTRUCT	logi	T	turns on/off interface reconstruction
ITIN_CONT_MAX	int	100	max. no. inner continuity iterations
ITIN_KE_MX	int	100	max. no. inner k or epsilon inner iterations
ITIN_SPC_MX	int	100	max. no. inner species iterations
ITIN_STRS_MAX	int	100	max. no. inner stress iterations
ITIN_TMF_MX	int	100	max. no. inner turbulence mass flux iterations
ITIN_TMP_MX	int	100	max. no. inner temperature iterations
ITIN_UVW_MX	int	100	max. no. inner velocity iterations
iup_rho	int	0	FOU density flag
iup_tmp	int	0	FOU temperature flag
iup_uvw	int	0	FOU velocity flag
matrix_solv	logi	F	PETSc flag
MAXCYC	int	99999998	max. no. cycles allowed
MDOTIN	real	0.00E+00	net mass flux crossing inflow boundary
MESH_MOTION	chad	euler	mesh movement type
MESHFILE		chad.msh	file name for mesh data
MIX	logi	F	indicates if mixing in computational zones is allowed
mon_x	int	0.00E+00	X-monitoring point
mon_y	int	0.00E+00	Y-monitoring point
mon_z	int	0.00E+00	Z-monitoring point
NC_DUMP	int	99999999	cycle frequency for writing restart dumps

Table E.5.1 Continued

NC_DUMP_STA RT	int	0	cycle after which NC_DUMP becomes active
NC_POST	int	99999999	cycle frequency for writing post dumps
NC_POST_STA RT	int	0	cycle after which NC_POST becomes active
NC_SEMILAGR ANGE	int	1	cycle frequency for semilagrange mesh motion
NC_TTY	int	1	cycle frequency for terminal output
NMATS	int	1	number of materials
NSPAREN	int	3	number of SPAREN arrays
NXIN	real	1.00E+00	X-component of the unit normal in the inflow direction
NXPISTON	real	0.00E+00	X-component of the unit normal pointing to piston movement
NYIN	real	0.00E+00	Y-component of the unit normal in the inflow direction
NYPISTON	real	0.00E+00	Y-component of the unit normal pointing to piston movement
NZIN	real	0.00E+00	Z-component of the unit normal in the inflow direction
NZPISTON	real	1.00E+00	Z-component of the unit normal pointing to piston movement
OUT_TYPE	int	1	inflow/outflow boundary type
OUTAVS	logi	F	flag for writing AVS post dump
OUTENS	logi	F	flag for writing Enscript post dump
OUTFV	logi	F	flag for writing Fieldview post dump
OUTGMV	logi	T	flag for writing GMV post dump
OUTPUT_CONT ROLS	char	uninitializ ed	contains switches for controlling all output

Table E.5.1 Continued

PARTICLE_MODEL	char	none	particle model type
PGSMAX	real	hugenum	maximum allowable value of PGS
PGSMIN	real	1.00E+00	minimum allowable value of PGS
PGSRAT	real	0.04	maximum allowable ration of pressure fluctuations to mean pressure
PGSSTART	real	1.00E+00	initial value of of factor for scaling sound speed for PGS method
PIN	real	1.01E+05	inflow reservoir pressure
POSTFILE	char	chad.pst	name of post file
POUT	real	1.01E+05	inflow/outflow boundary fluid pressure
RESTART	logi	F	indicates problem restart
RESTFILE	char	chad.rst	restart file name
RHOIN	real	1.00E+00	inflow reservoir density
RHOOUT	real	1.00E+00	inflow/outflow boundary fluid density
RPRART	real	1.00E+00	reciprocal of artificial Prandtl number
RSCART	real	1.00E+00	reciprocal of artifical Schmidt number
RUNTYPE	char	normal_wi th_interfac	type of run
SESFIL	char	chad.ses.p rs	Sesame file name
SOCIT	real	5.00E-01	factor for second order convection in time
SPC_TABLE	char	data/chad. spc	species table filename
STEADY	logi	F	steady-state flag
STROKE	real	1.00E-01	piston travel length
TABULAR_EOS_OPTION	char	sesame_di rect	table lookup option for tabular equation of state

Table E.5.1 Continued

TESTRUN	char	none	name of test problem
THICKMIN	real	tinynum	approximate minimum region thickness
TIME0	real	0.00E+00	initial problem startup time
TIN	real	3.00E+02	inflow reservoir temperature
TKEIN	real	0.00E+00	inflow turbulent kinetic energy
TKEOUT	real	0.00E+00	inflow/outflow turbulent kinetic energy
TKESRC	real	0.00E+00	turbulent kinetic energy at k-source
TMAX	real	0.99*huge num	max. temperature
TMFIN	real	0.00E+00	inflow-boundary turbulent mass flux magnitude
TMFXOUT	real	0.00E+00	X-comp of turbulent mass flux for inflow-outflow turb. mass flux specified nodes
TMFXSRC	real	0.00E+00	X-comp. of turbulent mass flux for turb. mass flux specified nodes
TMFYOUT	real	0.00E+00	Y-comp of turbulent mass flux for inflow-outflow turb. mass flux specified nodes
TMFYSRC	real	0.00E+00	Y-comp. of turbulent mass flux for turb. mass flux specified nodes
TMFZOUT	real	0.00E+00	Z-comp of turbulent mass flux for inflow-outflow turb. mass flux specified nodes
TMFZSRC	real	0.00E+00	Z-comp. of turbulent mass flux for turb. mass flux specified nodes
TOUT	real	3.00E+02	inflow/outflow fluid temp for boundary or stagnation
TSRC	real	3.00E+02	temp. for temp specified nodes
TW	real	3.00E+02	wall temp. for computing heat transfer coefficient
USRC	real	0.00E+00	X-component of velocity at velocity specified nodes
VELIN	real	0.00E+00	net fluid speed at inflow boundaries
VERBOSITY		0	TTY output verbosity level

VISRAT		-6.67E-01	negative of lamda/mu ratio
VSRC	real	0.00E+00	Y-component of velocity at velocity specified nodes
WSRC	real	0.00E+00	Z-component of velocity at velocity specified nodes

A rather large number of variables created in CHAD_ANL has been omitted from this list. Because some portions of CHAD_ANL are not included in the present work their inclusion of the variables in this list would be meaningless.

E.6 Boundary Conditions in CHAD

CHAD offers the following boundary condition types: wall, inflow, inflow/outflow, symmetric, periodic, and interface. Walls, inflow, and inflow/outflow boundaries are briefly described in CHAD's physics manual, and are briefly reviewed here. Symmetric, periodic, and interface boundaries are not described in the physics manual. Boundary conditions are set principally via nodetypes, CONTAB, and the modifications to user FORTRAN routines. A list of user FORTRAN routines and their uses is described later. Summaries of the boundary condition options in CHAD are provided below, and are pieced together from portions of the physics manual, comments found within the code, and author experience.

E.6.1 Wall Boundaries

E.6.1.1 The Definition of a Wall

CHAD applies default logic, changeable in the routine FLAG_WALL, when determining whether or not a node is considered a wall node. The logic involves two parts. The first part computes a wall on all interface boundaries. This involves a check

against the `IPARENT` array. Since this array is used to determine parent nodes at interfaces and is also used when computing periodic boundary conditions, it only flags a node as a wall when the periodic option is not specified. The periodic and interface options may not be used in the same problem.

The second part of the logic checks the closed areas of a node. Any node with a non-negligible area exposed to the outside of the problem domain is determined to be a wall node. Node component areas are stored in the arrays `NXN_CLOSED`, `NYN_CLOSED`, and `NZN_CLOSED`. If the sum of the nodal values in these arrays is greater than `smallnum`, then the wall is flagged as a node.

The above definitions for determining walls involve caveats and cautions that need discussed. The first caution is that the modeler needs to be aware that `nodetype` is not checked when determining a wall. Therefore, interior, no-slip nodes would not resolve as walls. The obvious limitation therefore being that by default, CHAD does not allow for the inclusion of interior walls in a problem, such as a thin plate.

A second caution involves 2-D problems. In symmetric problems of this type, all nodes in the domain will compute as wall nodes, as all (symmetric) nodes have closed areas aligned along the symmetry plane. This is acceptable except in the case of turbulent flows. When 2-D turbulent flows are being modeled, this logic requires that the distance to the wall, `DISTWALL`, is greater than `hugenum` (typically, `hugenum+hugenum`) at non-wall nodes in the user subroutine `User_wall`.

The final consideration in defining a wall involves the CONTAB array. Since CHAD does not model boundary conditions based on faces, nodes that reside at the intersection of multiple boundary conditions, such as a wall and an inflow boundary, require satisfaction of multiple boundary conditions. In these cases, it is important that the CONTAB at these nodes be correct. If the CONTAB at these nodes is set to unity, no closed areas will be computed and the node will not be marked as a wall node.

E.6.1.2 Velocity Constraints in Laminar Flow

When nodes are specified as no-slip, velocities at the nodes are set to the velocity of the wall (UWALL, VWALL, and WWALL). When nodes are specified as free-slip, the velocity at the wall is computed using the finite difference approximation to the momentum equation with $\mathbf{u} \cdot \mathbf{n} = \mathbf{u}_{\text{wall}} \cdot \mathbf{n}$, where \mathbf{n} is the unit normal to the wall. A zero tangential stress on the wall is assumed.

E.6.1.3 Velocity Constraints in Turbulent Flow

In turbulent flows, the nodetype of a wall node is ignored, with the exception that it is incidentally used to determine the closed areas that define a wall. Momentum, enthalpy and K -equations are solved at the wall, with $\mathbf{u} \cdot \mathbf{n} = \mathbf{u}_{\text{wall}} \cdot \mathbf{n}$, where \mathbf{n} is the unit normal to the wall. Wall heat loss, shear stress, and boundary conditions on K and ε are determined using wall functions, the definitions of which may be found in the physics manual.

E.6.1.4 Dirichlet Boundary Conditions for Nodal Temperatures

By default, all walls in CHAD are set to a default wall temperature, TW , provided in the input file. If all walls are desired to be a single temperature, supplying TW as part of the input file is sufficient. When variable wall temperatures or multiple wall temperatures are desired, a value may be specified in either of two locations. The first and is in the subroutine `User_wall`, where the wall temperature ($TWALL$) may be provided. This method works as the value of $TWALL$ is ultimately used to set $TSOURCE$ which is used to prevent this node from changing in the solver routines. It, however, has the drawback of altering $TWALL$ which is used to compute heat loss through a node.

The second and probably more desired way of setting wall temperatures is to directly set $TSOURCE$ in the subroutine `User_srcs`. Again, this works the same way as described earlier, in that it prevents changes in temperature from occurring at temperature specified nodes during the solution routines.

E.6.1.5 Neumann Boundary Conditions for Nodal Temperatures

Adiabatic walls are specified in CHAD by specifying the heat transfer coefficient (HTC) to be zero. This is accomplished through user defined coding in the subroutine `User_wall`.

E.6.2 Inflow Boundaries

Inflow boundaries may take the form of either specified velocity, or specified mass flux. At these boundaries, temperature must be specified. Pressure will be computed from the continuity equation, and density will be computed from temperature and pressure. A fairly complete description of inflow boundaries is provided in the physics manual, but the variables and values required are not described. They are summarized in Table E.6.1

Table E.6.1: Inflow boundary variables

Parameter	Code Variables	Values
Inflow velocity, \mathbf{u}_{in} relative to normal wall motion	VELIN NXIN NYIN NZIN	Net fluid speed at inflow boundary, when IN_TYPE is 1 or 2 X-component of the unit normal in the the inflow direction Y-component of the unit normal in the the inflow direction Z-component of the unit normal in the the inflow direction
Reservoir pressure	pin	Reservoir pressure
Inflow temperature	tin	Reservoir temperature
Relationship of inflow to reference conditions	IN_TYPE	1. Inflow velocity specified; isothermal compression/expansion from reservoir condition to the inlet condition 2. Inflow velocity specified; isentropic compression /expansion from reservoir condition to the inlet condition 3. Inflow mass flux specified; isothermal compression /expansion from reservoir condition to the inlet condition 4. Inflow mass flux specified; isentropic compression /expansion from reservoir condition to the inlet condition
Inflow turbulent kinetic energy and dissipation rate, K_{in} and ε_{in}	tkein epsin	Value of turbulent kinetic energy at the inflow boundary Value of turbulent dissipation rate at the inflow boundary
Inflow mass fractions	MFRACIN	Array containing inflow mass fractions

If multiple inflow boundaries exist with different boundary conditions, user FORTRAN is required to individually set the specified boundary conditions.

E.6.3 Inflow/Outflow Boundaries

Inflow/outflow boundaries may assume either of two forms. The first role is specified hydrostatic pressure, while the second is a specified stagnation condition. The variable `out_type`, read from the input file, determines the type. If the hydrostatic pressure boundary is specified, only the pressure needs to be provided as `pout`. When a stagnation boundary is specified, a complete description of the flow at the outflow boundary is required.

E.6.2: Outflow boundary variables

Parameter	Code Variables	Values
Inflow/outflow boundary type	OUT_TYPE	<ol style="list-style-type: none"> 1. Specified static pressure at inflow/outflow boundary 2. Specified stagnation condition at inflow/outflow boundary
Hydrostatic pressure	pout	Reservoir pressure
Inflow temperature	tout	Reservoir temperature
Inflow turbulent kinetic energy and dissipation rate, K_{in} and ε_{in}	tkein epsin	Value of turbulent kinetic energy at the inflow boundary Value of turbulent dissipation rate at the inflow boundary
Inflow mass fractions	MFRACIN	Array containing inflow mass fractions

As with inflow boundaries, if multiple inflow boundaries exist with different boundary conditions, user FORTRAN is required to individually set the specified boundary conditions.

E.6.3 Symmetry Boundaries

Symmetry (reflective) boundary conditions exist in CHAD when any boundary that is at least partly closed is NOT and is not identified with a no-slip node. At reflective walls, the components of scalar gradients normal to the wall are zeroed. For vectors, appropriate off-diagonal terms of a vector are zeroed after rotating the vector into a coordinate system whose x -coordinate is normal to the surface. The user may modify this logic in the user FORTRAN routine `FLAG_REFL`.

By default, CHAD attempts to identify symmetric geometries at run-time in the routine `init_flags`. First, it will check for user-specified dimensionality given by the input file keyword `GEOMETRY`. Allowable values are 1-D tube, 1-D cone, 2-D plane, or 2-D wedge. If `GEOMETRY` has not been specified, CHAD attempts to identify the domain as one of the recognizable forms listed above. If any of the above forms is specified or recognized, CHAD identifies zero-flux connections and flags them in the array `SYMMETRYM`.

In the computation of cell-face velocities, the `SYMMETRYM` flag is used to enforce a zero flux across connections between symmetric nodes. When the mesh motion is not purely Lagrangian, `SYMMETRYM` is copied into the array `NOFLUXM`, which is then used

to zero out any cell-face velocities. Since the volumetric flow rate is computed using the cell-face velocities, the volumetric flow rate between cells will also be zero.

Lower dimensional problems, such as 2-D plane, are not truly of a lower dimension. For example, a volume does not reduce to a surface, nor does a surface reduce to an edge in CHAD. CHAD is a fully 3-D software code. Therefore, a 2-D planar problem will in fact have two separate planes of nodes, and two separate control volumes associated with each node for each plane. As a result, it is possible for the values of nodes opposite of each other on a symmetry plane to acquire different values. In such situations, CHAD computes the average value between the two nodes, and applies it as the nodal value for each node.

E.6.4 Interface and Periodic Boundaries

Neither interface nor periodic boundaries are described completely within the physics manual or inside the code. Additionally, they have not been used by this author within the course of this work, so an authoritative description of these boundary conditions is not provided.

E.7 Running CHAD

Executing CHAD is a straightforward process both in parallel and in serial forms. CHAD looks for a `chad.in` file in the path given in the input file (or the current working directory if no explicit path is given) at the time of execution. If the file is not

found, CHAD will stop. The input file, `chad.in`, specifies all other files required, such as the mesh file and a restart file if required.

CHAD does not write an output file containing data summaries and progress while executing, so it is a good idea to redirect the output of a CHAD simulation into a file so that you may store a record of the simulation progress, especially in the case that something should not work properly. To do this simply use the redirect and output to a specified file name, such as `chad.x > chad.out`. Depending on the shell being used, the syntax for redirecting output may be slightly different.

There are a few things to keep in mind when you run CHAD. By default CHAD's convergence criteria specify that CHAD will almost always execute until a maximum time or number of iterations or time is reached. Therefore, expect to control your simulation through the terms `MAXCYC` or `TMAX`, which control the maximum number of pseudo time steps or maximum time. In this dissertation, the ability to control CHAD through residual values has been added and may be used as an alternate way of terminating CHAD.

If you are debugging CHAD with the Absoft compilers, use the `xfx` debugger and restart from a converged solution or an analytic solution. Use the user subroutines to specify known variable fields before you begin. Run parallel simulations in serial before you begin. Serial jobs are far easier to debug than parallel jobs (especially since the Absoft debugger `xfx` cannot debug parallel jobs).

E.8 Post-Processing CHAD

We have exclusively used the Generalized Mesh Viewer (GMV) available through Sandia National Labs to post process CHAD simulation results. GMV files are output when the variable `outgmw` is specified. The name of the output file is read from the input file and stored in the variable `POSTFILE`.

Output files are written at the start of a simulation, and at time intervals controlled by `DT_POST_START`, `DT_POST`, with an accuracy given by `DT_POST_ACCURACY`, as well as by the number of time steps performed (cycles) via `NC_POST`, and `NC_POST_START`. The data written to post files and during a simulation is summarized in Table E8.1.

E.8.1: Data Written to Post-Processing Files

Variables at t = 0	Variables at t > 0	Description
BB	BB	
BT		burn time of a reactive material
CONTAB(1, :)		boundary constraint table for index (xx)
CONTAB(2, :)		boundary constraint table for index (xy)
CONTAB(3, :)		boundary constraint table for index (xz)
CONTAB(4, :)		boundary constraint table for index (yy)
CONTAB(5, :)		boundary constraint table for index (yz)
CONTAB(6, :)		boundary constraint table for index (zz)
EPS	EPS	turbulent energy dissipation
I	I	specific internal energy
ICOLORE		processor id on which the element lies
ICOLORN		processor id on which the node lies
LIQDENS*	LIQDENS*	liquid macroscopic density for spray models
MASSFRAC*	MASSFRAC*	mass fraction
NODETYPE		node type
NX_CLOSE		x-component of unit vector normal to closed boundary pointing inwards (into the zone)

Variables at t = 0	Variables at t > 0	Description
NY_CLOSE		y-component of unit vector normal to closed boundary pointing inwards (into the zone)
NZ_CLOSE		z-component of unit vector normal to closed boundary pointing inwards (into the zone)
OPEN_BDRY		open boundary flag
P	P	nodal pressure
REFLECT		reflective boundary flag
REGNOE		region number associated with an element
REGNON		region number associated with a node
RHO	RHO	nodal density
ROTXX*	ROTXX*	xx-component of rigid body rotation tensor
ROTXY*	ROTXY*	xy-component of rigid body rotation tensor
ROTXZ*	ROTXZ*	xz-component of rigid body rotation tensor
ROTYX*	ROTYX*	yx-component of rigid body rotation tensor
ROTYZ*	ROTYZ*	yz-component of rigid body rotation tensor
ROTYZ*	ROTYZ*	yz-component of rigid body rotation tensor
ROTZX*	ROTZX*	zx-component of rigid body rotation tensor
ROTZY*	ROTZY*	zy-component of rigid body rotation tensor
ROTZZ*	ROTZZ*	zz-component of rigid body rotation tensor
SMR*	SMR*	droplet Sauter mean radius for spray model
SPAREN(1, :)	SPAREN(1, :)	spare number array 1
SPAREN(2, :)	SPAREN(2, :)	spare number array 2
SPAREN(3, :)	SPAREN(3, :)	spare number array 3
T	T	nodal temperature
TKE	TKE	turbulent kinetic energy
TMFX*	TMFX*	x-component of the turbulence mass flux
TMFY*	TMFY*	y-component of the turbulence mass flux
TMFZ*	TMFZ*	z-component of the turbulence mass flux
U	U	x-component of velocity
V	V	y-component of velocity
VOLFRAC*	VOLFRAC*	volume fraction
W	W	z-component of velocity
X	X	nodal x coordinate
X-VORT	X-VORT	xcomponent of vorticity
Y	Y	nodal x coordinate
Y-VORT	Y-VORT	y-component of vorticity
Z	Z	nodal z coordinate
Z-VORT	Z-VORT	z-component of vorticity
* Available only when associated physics packages are used.		

If the user wishes to post-process data other than those available in the default post files, the `SPAREN` arrays may be used for nodal data, or, custom post-processing data may be written through the user FORTRAN routine `User_post`.

E.9 Internals of CHAD

It is important to understand the structure of CHAD so that when modifying CHAD, as is necessary with setting boundary conditions and customizing features, a larger picture of the overall interaction with the code is available.

E.9.1 Compiler Directives

During the compilation stage of CHAD, compiler directives are used to select portions of the code for use. For example, when a parallel compilation for CHAD is specified, parallel communication routines are selected for transferring data within the code as opposed to the default which uses internal communication. Similarly, when the pressure-based version of the code is desired, density based portions of the code are excluded. These compiler options are specified in the makefile.

In the installation performed at ANL, the compiler directives `-DPARALLEL` and `-DUSE_PGSLIB` are used to compile CHAD in parallel using the PGSLib communications library. Any code within CHAD contained within compiler these compiler directives will automatically be kept in the preprocessing step from `.FF` to `.F` files. Similarly, the compiler directive for the pressure-based version is specified using `-`

DPRESSURE_BASED. Finally, the SIMPLE algorithm is chosen using the compiler flag `-DSIMPLE`.

If High Performance Computing is specified all HPF compiler directives are modified through definitions found in the header file `MACROS.HH`. If a “Connection Machine 5” is being used, directives for CM5 are specified. When neither HPF nor CM5, are desired, these directives are translated into comments.

E.9.2 Data Types

Numeric data types in CHAD are parameterized using the FORTRAN `kind` statement. These statements allow for the explicit declaration of a variable length data type, allowing for portability across machines.

Several custom data types are also defined in CHAD. One of the most useful types may be the `PEInfo` type which contains `NPE` as the number of processors, `thisPE` for the rank of this PE, the `IO_PE` which is the rank of the PE doing IO and the `IOP` which is a logical flag indentifying the input/output processor. The `IOP` and `thisPE` are useful for debugging parallel jobs as `thisPE` easily identifies which processor a variable resides on while `IOP` allows quick and dirty print statements to the TTY without going through CHADS internal method for writing out data.

E.9.3 Parallel Communication

Parallel communication within CHAD is centralized within `Parallel_module_c.FF`. Within this module, variables, types, and procedures are defined for a variety of parallel computing environments.

Due to the unstructured mesh capability of CHAD, communication between nodes, connections, and elements is computing using gather and scatter operations. Gather operations involve acquiring information from surrounding nodes or connections, while scatter operations involve sending information to adjacent nodes or connections. Knowing how to get and provide information when needed and knowing which function to call is important. There are a total of 36 gather/scatter operations available in CHAD and they are summarized in Table E.9.1

Table E.9.1: Gather/Scatter operations available in CHAD.

Gather Operations	Type of Data
<i>Gathers type of data from connection mid-points to element edges</i>	
<code>gatherb_sm</code>	scalar
<code>gatherb_v6m</code>	6-component vector
<code>gatherb_v9m</code>	9-component vector
<code>gatherb_vm</code>	vector
<i>Gathers type of data from parent to children nodes</i>	
<code>gatherp_sc</code>	scalar
<code>gatherp_v6c</code>	6-component vector
<code>gatherp_v9c</code>	9-component vector
<code>gatherp_vc</code>	vector
<i>Gathers type of data from nodes to connection terminuses</i>	
<code>gathert_sn</code>	scalar
<code>gathert_v6n</code>	6-component vector
<code>gathert_v9n</code>	9-component vector
<code>gathert_vn</code>	vector
<i>Gathers type of data from nodes to element vertices</i>	
<code>gatherv_sn</code>	scalar
<code>gatherv_v6n</code>	6-component vector
<code>gatherv_v9n</code>	9-component vector

Gather Operations	Type of Data
<code>gather_vn</code>	vector
Scatter Operations	Type of Data
<i>Scatters type of data from element edges to connection mid-points</i>	
<code>scatterb_sm</code>	scalar
<code>scatterb_v6m</code>	6-component vector
<code>scatterb_v9m</code>	9-component vector
<code>scatterb_vm</code>	vector
<i>Scatters type of data from connection mid-points to nodes</i>	
<code>scatterm_sn</code>	scalar
<code>scatterm_v6n</code>	6-component vector
<code>scatterm_v9n</code>	9-component vector
<code>scatterm_vn</code>	vector
<i>Scatters type of data from children nodes to parents</i>	
<code>scatterp_sc</code>	scalar
<code>scatterp_v6c</code>	6-component vector
<code>scatterp_v9c</code>	9-component vector
<code>scatterp_vc</code>	vector
<i>Scatters type of data from connection terminuses to nodes</i>	
<code>scatterv_sn</code>	Scalar
<code>scatterv_v6n</code>	6-component vector
<code>scatterv_v9n</code>	9-component vector
<code>scatterv_vn</code>	Vector

E.9.4 CHAD Execution Sequence

CHAD execution begins with the driving routine `chad_main`. Within this subroutine, the first major code execution is parallel initialization which occurs in the routine `parallel_init`. Next, the mesh is read in during a call to `read_mesh`. After the mesh is read, the input file is read in `read_input`. Any data not found in the input file is assigned a default value contained within that routine. At this point, CHAD calls `driver` which essentially transfers control to `cycle_loop`. This subroutine serves the primary purpose of terminating the code based upon any of the following criteria.

- ♦ If the number of cycles (`ncyc`) exceeds the maximum allowed (`maxcyc`)
- ♦ If crank angle (`crank`) exceeds the maximum allowed (`camax`)
- ♦ If the problem time (`time`) exceeds the maximum allowed (`tmax`)
- ♦ If it is not (`converged`), meaning time step advancement is not possible
- ♦ If there is a problem computing the time step, resulting in a `timstp_error` flag
- ♦ If the maximum cpu time is exceeded (`cpu_max`)

Unlike many commercial CFD codes, CHAD does not halt execution based upon a specified decrease in the residuals of the fundamental equations governing the system. This is because CHAD did not originate as a CFD code and traditional CFD code stopping criterion are not applicable to many of the problems CHAD is capable of solving.

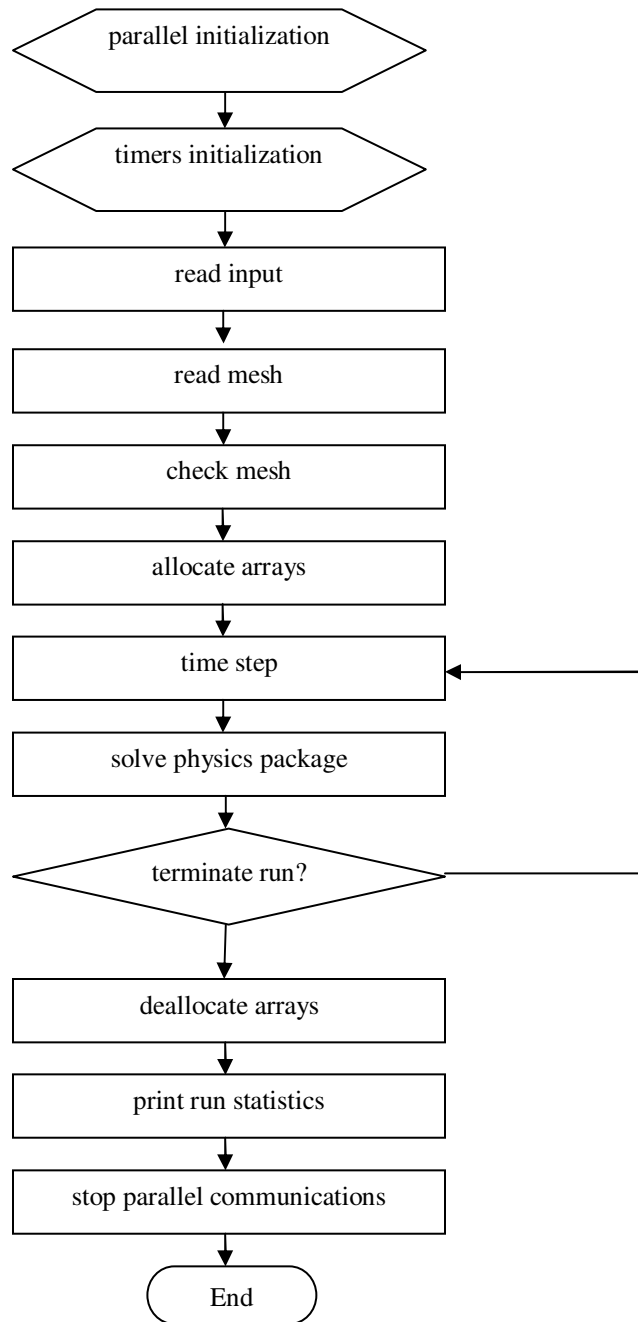


Figure E.9.1: Solution flow diagram for CHAD.

E.9.5 Physics Routines

The Physics Packages

Within `physics`, the governing sets of equations for the different physics packages are solved. It begins by initializing boundary quantities. Then, it calls the driver for the particle spray model (when activated.) Next, it solves the hydrodynamic routines (momentum, energy, and continuity. It is here that it differentiates between solution methods when explicit and non-explicit solution methods are used. After the hydrodynamic routines, it finishes some parts of the spray model, and then it calls for the equation of state to be solved. It concludes by calling for the solution to the k-epsilon turbulence equations..

E.9.6 Hydrodynamic Routines

The solution portion for the Navier-Stokes equations is primarily controlled in `hydro`. `Hydro` solves the Navier-Stokes equations using either the SIMPLE or NEWTON-KRYLOV methods. As only the SIMPLE method is used within the scope of this thesis the NEWTON-KRYLOV method will be ignored

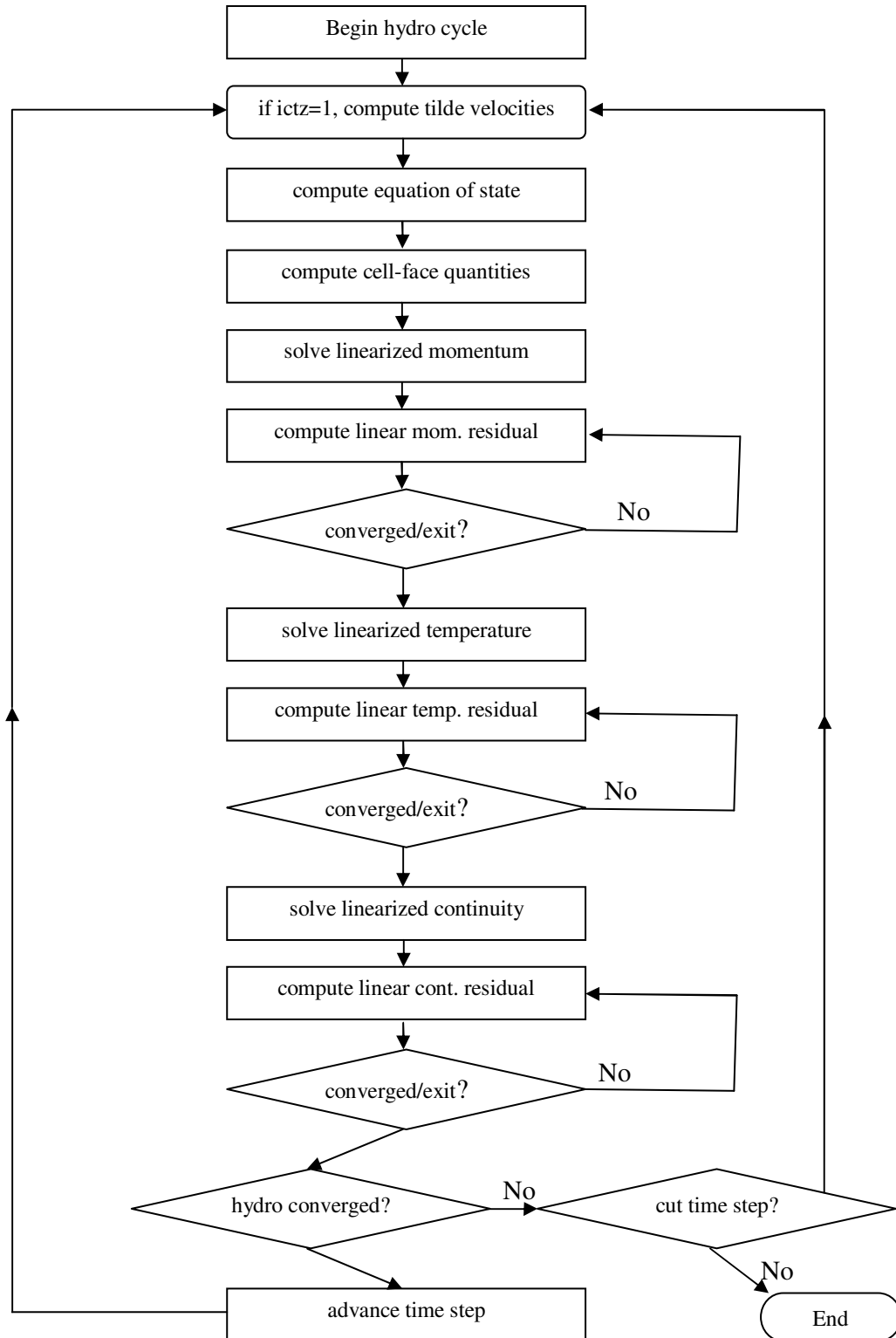


Figure E.9.2: Flow diagram for the hydrodynamic solution algorithm.

E.9.7 Time Stepping

If the code is not stopped by any of the previously listed criteria, `transient` is called to advance the time step. For a detailed explanation of the time stepping procedures for CHAD the user is advised to consult the CHAD physics manual as it is beyond the scope of this User's Guide. However, let it be sufficient to say that within `transient`, the physics package routines are called, specifically the hydrodynamic equations, the species equations, and the turbulence equations. Generally, `transient` makes a single call to `physics` if time-step advancement is possible, otherwise, it prepares to abort the code.

E.9.7 Iterative Controls

The iterative controls to CHAD's SIMPLE method are found in the subroutine `hydro`. These controls are altered during the solution process based upon the convergence properties of the inner iterations. However, there are restrictions on the number of outer iterations and they are listed in Table E.9.2

CHAD performs such a calculation when testing nodes for frictionless walls.

Table E.9.2: Iterative Limits for the CHAD Simple Algorithm

Variable	Meaning	Value
<code>iterin_mx</code>	maximum # outer iterations	20
<code>iterout_use</code>	number of outer iterations to use	<code>iterout_si</code> for semi-implicit
<code>iterout_si</code>	number of iterations to aim for during the semiimplicit mode	3 for semi-implicit ~10 for steady flow
<code>limit_semi-implicit</code>	flag which determines when to switch to fully implicit mode	$\min(\text{iterout_mx}/2, 2 * \text{iterout_si})$
<code>iterout</code>	current outer iteration number	current outer iteration number

Outer Iteration Convergence Checks

The following convergence checks are performed in the outer iteration before the solution of the Navier-Stokes equations but after the computation of the equation of state. If any node's continuity residual exceeds 75% of its density, or if the nodal temperature of any node is less than $-1/2$ (yes, negative temperatures are allowed for special cases not described here) the data is flagged as 'bad'. Continue.

If the current outer iteration is one less than the number of outer iterations to be used, and the number of outer iterations to use is less than the maximum number of outer iterations, and the maximum of any of the inner linearized iteration counts exceeds $20 \cdot \min(\text{iterout}, 4)$, OR 'bad' data was flagged in the previous step, then increment the number of allowed outer iterations by one.

```
*** Increments allowable iteration count BEFORE they
"solve" routines have a chance to follow the last
outer-iteration path in case of semi-implicit or
steady-state case.
```

If the outer iteration count is equal to the maximum number allowed, and the number of outer iterations to be used is less than the maximum number allowed, and the data is 'bad', prepare to reduce the time step.

```
*** If it was the last outer iteration for semi-
implicit or steady-state case and we still have
problems with the data (whereas we did not have
problems with the data during the previous outer
```

iteration -- because the test above this test passed during the previous iteration), it is too late to do anything because we already have followed a special path in the "solve" routines. Therefore, get ready to reduce the time step.

If the current iteration count is less than the desired outer iteration count, and the number of outer iterations to use is less than the maximum number of outer iterations and the current iteration count is greater than the limit of the semi-implicit mode, then set the maximum number of outer iterations to be used as the maximum number of allowable outer iterations.

*** If the semi-implicit or steady-state option has not converged within a specified limit, something is gone wrong. To prevent false convergence, we essentially switch to fully implicit option in this case. However, if this is the last outer iteration (iterout=iterout_use), it is too late to do anything because we already have followed a special path in the "solve" routines. In this case we proceed normally instead of reducing the time step because the limit is subjective and has enough safety factor built into this count.

If the outer iteration is flagged as converged, or unable to be converged in an inner iteration or the current outer iteration count is equal to or exceeds the number of outer iterations to use, then check to see if both the equation of state and continuity equations converged and the its steady or semi-implicit and the outer iteration count is less than the semi-implicit limit. In these cases, flag the outer iteration as converged and exit the SIMPLE cycle.

*** For the semi-implicit option, we want to flag convergence only if the number of outer iterations was reasonable. Otherwise, we want to make sure the solution was fully converged (inner iterations during the last outer iteration were all one). If we do not do this, there is a danger of false convergence.

Appendix F: Source Code Modifications

A variety of files within CHAD have been added or changed in order to implement the reformulated continuity equation, PETSc, and tools discussed in this dissertation.. Those changes are described within this appendix.

Because CHAD is an export controlled piece of software, the full source code is not provided within this dissertation (indeed, there would be too much to print anyway.) Therefore for each subroutine a brief description will be provided to familiarize the reader with what the changes encompass. As was mentioned early in this dissertation, the structure of CHAD_ANL corresponded to approximately CHAD 3.0 and was different than CHAD 5.0. A difference of the source listings is provided between CHAD_ANL and CHAD_UIUC is provided. The individual routine changes made in association with the migration of CHAD_ANL into CHAD 5.0 and CHAD_UIUC are not presented here. However, the individual routine differences between the migrated version of CHAD_ANL/CHAD 5.0 and CHAD_UIUC is.

Each change will list its relevance to other changes made (particularly if they apply to PETSc or the reformulated continuity equation.) In the event a subroutine is modified the changes will simply be discussed. In the case of subroutines which have been added, they will be supplied in their entirety. The routines following are listed in alphabetical order.

F.1 Diff of CHAD_ANL to CHAD_UIUC

Older file -> C:\chad_anl\anl_data_module_c.FF *than*
C:\chad_uiuc\anl_data_module_c.FF
Only in (second) C:\chad_uiuc -> anti_diff_mats.FF
Only in (second) C:\chad_uiuc -> api.FF
Only in (second) C:\chad_uiuc -> api_module_c.FF
Older file -> C:\chad_anl\arrays_glo_module_c.FF *than*
C:\chad_uiuc\arrays_glo_module_c.FF
Only in (first) C:\chad_anl -> arrays_glo_module_c.orig
Only in (second) C:\chad_uiuc -> assoc_flow.FF
Older file -> C:\chad_anl\bcd_data_module_c.FF *than*
C:\chad_uiuc\bcd_data_module_c.FF
Older file -> C:\chad_anl\bcs.FF *than* C:\chad_uiuc\bcs.FF
Only in (first) C:\chad_anl -> bcs.orig
Older file -> C:\chad_anl\bcs_dep_module_d.FF *than*
C:\chad_uiuc\bcs_dep_module_d.FF
Only in (second) C:\chad_uiuc -> burn_fracs_prog.FF
Only in (second) C:\chad_uiuc -> burn_fracs_react.FF
Only in (second) C:\chad_uiuc -> burn_intervals.FF
Only in (first) C:\chad_anl -> cellface_module_d.CTZ
Older file -> C:\chad_anl\cellface_module_d.FF *than*
C:\chad_uiuc\cellface_module_d.FF
Only in (first) C:\chad_anl -> cellface_module_d.orig
Only in (first) C:\chad_anl -> CHAD.FF
Only in (second) C:\chad_uiuc -> chad_main.FF
Only in (second) C:\chad_uiuc -> CHAD_P.FF
Older file -> C:\chad_anl\com_setup.FF *than*
C:\chad_uiuc\com_setup.FF
Only in (first) C:\chad_anl -> commdata_module_c.FF
Only in (second) C:\chad_uiuc -> conserve.FF
Only in (first) C:\chad_anl -> conserve.orig
Only in (second) C:\chad_uiuc -> conserve_mats.FF
Older file -> C:\chad_anl\constants_module_c.FF *than*
C:\chad_uiuc\constants_module_c.FF
Only in (second) C:\chad_uiuc -> construct_eletype.FF
Only in (first) C:\chad_anl -> curvature
Only in (second) C:\chad_uiuc -> cycle_loop.FF
Older file -> C:\chad_anl\dante_int_module_d.FF *than*
C:\chad_uiuc\dante_int_module_d.FF
Only in (second) C:\chad_uiuc -> debug-print.FF
Only in (first) C:\chad_anl -> DEFINE.h
Only in (first) C:\chad_anl -> DEFINE.HH

```

Older      file      ->      C:\chad_anl\deltas.FF      *than*
C:\chad_uiuc\deltas.FF
Older file -> C:\chad_anl\dimensions_module_c.FF *than*
C:\chad_uiuc\dimensions_module_c.FF
Older file -> C:\chad_anl\dissip_n_module_d.FF *than*
C:\chad_uiuc\dissip_n_module_d.FF
Older file -> C:\chad_anl\distribute_mesh_module_d.FF
*than* C:\chad_uiuc\distribute_mesh_module_d.FF
Older      file      ->      C:\chad_anl\driver.FF      *than*
C:\chad_uiuc\driver.FF
Only in (first) C:\chad_anl -> driver.orig
Only in (second) C:\chad_uiuc -> eigen.FF
Older file -> C:\chad_anl\element_module_c.FF *than*
C:\chad_uiuc\element_module_c.FF
Older      file      ->      C:\chad_anl\enthdiff.FF      *than*
C:\chad_uiuc\enthdiff.FF
Only in (second) C:\chad_uiuc -> enthflux.FF
Older      file      ->      C:\chad_anl\eos_module_d.FF      *than*
C:\chad_uiuc\eos_module_d.FF
Only in (first) C:\chad_anl -> eos_module_d.orig
Only in (second) C:\chad_uiuc -> eos_ses_module_d.FF
Older file -> C:\chad_anl\eosdata_module_c.FF *than*
C:\chad_uiuc\eosdata_module_c.FF
Only in (first) C:\chad_anl -> eosdata_module_c.orig
Only in (second) C:\chad_uiuc -> eospac_tables.FF
Only in (second) C:\chad_uiuc -> error_check_module_d.FF
Only in (first) C:\chad_anl -> FF
Only in (second) C:\chad_uiuc -> flow_stress.FF
Only in (second) C:\chad_uiuc -> flux_corr.FF
Only in (first) C:\chad_anl -> fnew1.zip
Only in (first) C:\chad_anl -> fnew2.zip
Older file -> C:\chad_anl\gather_module_d.FF *than*
C:\chad_uiuc\gather_module_d.FF
Older file -> C:\chad_anl\grads_module_d.FF *than*
C:\chad_uiuc\grads_module_d.FF
Only in (first) C:\chad_anl -> hem_deltat.FF
Only in (first) C:\chad_anl -> hem_xmassv.FF
Older      file      ->      C:\chad_anl\hydro.FF      *than*
C:\chad_uiuc\hydro.FF
Only in (first) C:\chad_anl -> hydro.orig
Only in (second) C:\chad_uiuc -> hydro_exp.FF
Older file -> C:\chad_anl\hydro_module_c.FF *than*
C:\chad_uiuc\hydro_module_c.FF
Only in (second) C:\chad_uiuc -> ieflux.FF
Older      file      ->      C:\chad_anl\improv_solt.FF      *than*
C:\chad_uiuc\improv_solt.FF
Only in (second) C:\chad_uiuc -> improve_sol.FF

```

```

Only in (first) C:\chad_anl -> improve_sol.orig
Older file -> C:\chad_anl\init_flags.FF *than*
C:\chad_uiuc\init_flags.FF
Older file -> C:\chad_anl\init_mesh.FF *than*
C:\chad_uiuc\init_mesh.FF
Only in (second) C:\chad_uiuc -> inout_module_d.FF
Older file -> C:\chad_anl\inoutdata_module_c.FF *than*
C:\chad_uiuc\inoutdata_module_c.FF
Older file -> C:\chad_anl\intrfc_init.FF *than*
C:\chad_uiuc\intrfc_init.FF
Older file -> C:\chad_anl\intrfc_module_d.FF *than*
C:\chad_uiuc\intrfc_module_d.FF
Only in (second) C:\chad_uiuc -> intrfc_nullify.FF
Older file -> C:\chad_anl\iterdata_module_c.FF *than*
C:\chad_uiuc\iterdata_module_c.FF
Older file -> C:\chad_anl\ke_consts_module_c.FF *than*
C:\chad_uiuc\ke_consts_module_c.FF
Older file -> C:\chad_anl\kepsilon.FF *than*
C:\chad_uiuc\kepsilon.FF
Older file -> C:\chad_anl\kepsilon_module_c.FF *than*
C:\chad_uiuc\kepsilon_module_c.FF
Older file -> C:\chad_anl\Kinds_module_c.FF *than*
C:\chad_uiuc\Kinds_module_c.FF
Only in (first) C:\chad_anl -> lset_curvature.FF
Only in (first) C:\chad_anl -> lset_curvature.orig
Only in (first) C:\chad_anl -> lset_deltarho.FF
Only in (first) C:\chad_anl -> lset_density.orig
Only in (first) C:\chad_anl -> lset_denvisc.FF
Only in (first) C:\chad_anl -> macros.h
Only in (first) C:\chad_anl -> macros.HH
Only in (first) C:\chad_anl -> matrix1_data_module_c.FF
Only in (first) C:\chad_anl -> matrix1_load.FF
Only in (first) C:\chad_anl -> matrix1_setk.FF
Only in (first) C:\chad_anl -> matrix1_setup.FF
Only in (first) C:\chad_anl -> matrix1_ysmp.FF
Only in (first) C:\chad_anl -> matrix1_ysmp0.FF
Only in (first) C:\chad_anl ->
matrix1_ysmp_data_module_c.FF
Only in (second) C:\chad_uiuc -> matrix_coeffs.FF
Only in (second) C:\chad_uiuc -> mesh_connectivity.FF
Older file -> C:\chad_anl\mesh_conns.FF *than*
C:\chad_uiuc\mesh_conns.FF
Only in (second) C:\chad_uiuc -> mesh_conns_par.FF
Only in (second) C:\chad_uiuc -> mesh_consistency.FF
Only in (second) C:\chad_uiuc -> mesh_fix.FF
Older file -> C:\chad_anl\mesh_move.FF *than*
C:\chad_uiuc\mesh_move.FF

```

Only in (first) C:\chad_anl -> mesh_move_lag.FF
 Only in (second) C:\chad_uiuc -> mesh_move_lag_module_d.FF
 Older file -> C:\chad_anl\mesh_update.FF *than*
 C:\chad_uiuc\mesh_update.FF
 Only in (second) C:\chad_uiuc -> mesh_vel_only.FF
 Only in (second) C:\chad_uiuc -> misc_update.FF
 Older file -> C:\chad_anl\ncoupler.FF *than*
 C:\chad_uiuc\ncoupler.FF
 Older file -> C:\chad_anl\nodetypes_module_c.FF *than*
 C:\chad_uiuc\nodetypes_module_c.FF
 Older file -> C:\chad_anl\options_module_c.FF *than*
 C:\chad_uiuc\options_module_c.FF
 Older file -> C:\chad_anl\orthdx_s.FF *than*
 C:\chad_uiuc\orthdx_s.FF
 Older file -> C:\chad_anl\orthdx_v.FF *than*
 C:\chad_uiuc\orthdx_v.FF
 Older file -> C:\chad_anl\orthdx_v5.FF *than*
 C:\chad_uiuc\orthdx_v5.FF
 Older file -> C:\chad_anl\orthdx_v6.FF *than*
 C:\chad_uiuc\orthdx_v6.FF
 Only in (first) C:\chad_anl -> output_module_d.FF
 Only in (second) C:\chad_uiuc -> parallel_debug.FF
 Only in (first) C:\chad_anl -> Parallel_info_module_c.FF
 Only in (second) C:\chad_uiuc -> Parallel_module_c.FF
 Only in (first) C:\chad_anl -> parallel_util_module_d.FF
 Only in (first) C:\chad_anl -> part_mesh.FF
 Only in (second) C:\chad_uiuc -> part_mesh_module_d.FF
 Only in (second) C:\chad_uiuc -> petsc_chad_finalize.FF
 Only in (second) C:\chad_uiuc -> petsc_chad_init.FF
 Older file -> C:\chad_anl\physdata_module_c.FF *than*
 C:\chad_uiuc\physdata_module_c.FF
 Older file -> C:\chad_anl\physics.FF *than*
 C:\chad_uiuc\physics.FF
 Only in (first) C:\chad_anl -> physics.orig
 Older file -> C:\chad_anl\physics_module_c.FF *than*
 C:\chad_uiuc\physics_module_c.FF
 Older file -> C:\chad_anl\piston.FF *than*
 C:\chad_uiuc\piston.FF
 Older file -> C:\chad_anl\pistondata_module_c.FF *than*
 C:\chad_uiuc\pistondata_module_c.FF
 Only in (second) C:\chad_uiuc -> polar_decomp.FF
 Only in (second) C:\chad_uiuc -> precondition_module_d.FF
 Only in (second) C:\chad_uiuc -> prep.FF
 Only in (first) C:\chad_anl -> read_anl_input.FF
 Only in (second) C:\chad_uiuc -> read_arrays.FF
 Only in (first) C:\chad_anl -> read_hem.FF

```

Older      file      ->      C:\chad_anl\read_input.FF      *than*
C:\chad_uiuc\read_input.FF
Only in (first) C:\chad_anl -> read_input.orig
Older      file      ->      C:\chad_anl\read_mesh.FF      *than*
C:\chad_uiuc\read_mesh.FF
Only in (second) C:\chad_uiuc -> read_msh_or_rst_dist.FF
Only in (second) C:\chad_uiuc -> read_msh_or_rst_iope.FF
Older      file      ->      C:\chad_anl\read_rst.FF      *than*
C:\chad_uiuc\read_rst.FF
Older      file      ->      C:\chad_anl\read_spc.FF      *than*
C:\chad_uiuc\read_spc.FF
Older      file      ->      C:\chad_anl\readvars_module_d.FF  *than*
C:\chad_uiuc\readvars_module_d.FF
Only in (first) C:\chad_anl -> res_cont.FF
Only in (first) C:\chad_anl -> res_ke.FF
Only in (second) C:\chad_uiuc -> res_lin_cont.FF
Only in (second) C:\chad_uiuc -> res_lin_hydro.FF
Only in (second) C:\chad_uiuc -> res_lin_ke.FF
Only in (second) C:\chad_uiuc -> res_lin_mat.FF
Only in (second) C:\chad_uiuc -> res_lin_spc.FF
Only in (second) C:\chad_uiuc -> res_lin_strs.FF
Only in (second) C:\chad_uiuc -> res_lin_tmf.FF
Only in (second) C:\chad_uiuc -> res_lin_tmp.FF
Only in (second) C:\chad_uiuc -> res_lin_tmp_exp.FF
Only in (second) C:\chad_uiuc -> res_lin_uvw.FF
Only in (second) C:\chad_uiuc -> res_lin_uvw_exp.FF
Only in (second) C:\chad_uiuc -> res_lin_uvw_lag.FF
Only in (second) C:\chad_uiuc -> res_lin_xyz.FF
Only in (second) C:\chad_uiuc -> res_lin_xyz_avg.FF
Only in (second) C:\chad_uiuc -> res_lin_xyz_elp.FF
Only in (second) C:\chad_uiuc -> res_lin_xyz_elp_sor.FF
Only in (second) C:\chad_uiuc -> res_lin_xyz_rot.FF
Only in (second) C:\chad_uiuc -> res_lin_xyz_spc.FF
Only in (first) C:\chad_anl -> res_lset.FF
Only in (first) C:\chad_anl -> res_lset_reinitial.FF
Only in (first) C:\chad_anl -> res_lset_reinitial.orig
Only in (first) C:\chad_anl -> res_lset_reinitial.TZ
Only in (first) C:\chad_anl -> res_spc.FF
Only in (first) C:\chad_anl -> res_strs.FF
Only in (first) C:\chad_anl -> res_tmp.FF
Only in (first) C:\chad_anl -> res_uvw.FF
Only in (first) C:\chad_anl -> res_uvw_lag.FF
Only in (first) C:\chad_anl -> res_xyz.FF
Only in (second) C:\chad_uiuc -> run_statistics.FF
Older      file      ->      C:\chad_anl\scatter_module_d.FF  *than*
C:\chad_uiuc\scatter_module_d.FF
Only in (first) C:\chad_anl -> schad

```

Only in (second) C:\chad_uiuc -> ser_ary_transfer.FF
 Only in (second) C:\chad_uiuc -> set_units.FF
 Older file -> C:\chad_anl\shuffle_module_d.FF *than*
 C:\chad_uiuc\shuffle_module_d.FF
 Only in (second) C:\chad_uiuc -> solve_cg_v.FF
 Only in (first) C:\chad_anl -> solve_cont.FF
 Only in (first) C:\chad_anl -> solve_cont.jan2k
 Only in (first) C:\chad_anl -> solve_cont_matrix.FF
 Only in (first) C:\chad_anl -> solve_cont_matrix.orig
 Older file -> C:\chad_anl\solve_cont_module_c.FF *than*
 C:\chad_uiuc\solve_cont_module_c.FF
 Only in (second) C:\chad_uiuc -> solve_gcr_s.FF
 Only in (second) C:\chad_uiuc -> solve_gcr_t5.FF
 Only in (second) C:\chad_uiuc -> solve_gcr_v.FF
 Only in (second) C:\chad_uiuc -> solve_gcr_v6.FF
 Only in (second) C:\chad_uiuc -> solve_gcr_v_precond.FF
 Older file -> C:\chad_anl\solve_gmr_s.FF *than*
 C:\chad_uiuc\solve_gmr_s.FF
 Only in (second) C:\chad_uiuc -> solve_gmr_t5.FF
 Older file -> C:\chad_anl\solve_gmr_v.FF *than*
 C:\chad_uiuc\solve_gmr_v.FF
 Older file -> C:\chad_anl\solve_gmr_v6.FF *than*
 C:\chad_uiuc\solve_gmr_v6.FF
 Only in (second) C:\chad_uiuc -> solve_gmr_v_precond.FF
 Only in (second) C:\chad_uiuc -> solve_hydro.FF
 Only in (second) C:\chad_uiuc -> solve_hydro_exp.FF
 Only in (second) C:\chad_uiuc -> solve_hydro_module_c.FF
 Older file -> C:\chad_anl\solve_ke.FF *than*
 C:\chad_uiuc\solve_ke.FF
 Older file -> C:\chad_anl\solve_ke_module_c.FF *than*
 C:\chad_uiuc\solve_ke_module_c.FF
 Only in (first) C:\chad_anl -> solve_lset.FF
 Only in (first) C:\chad_anl -> solve_lset_module_c.FF
 Only in (first) C:\chad_anl -> solve_lset_reinitial.bakup
 Only in (first) C:\chad_anl -> solve_lset_reinitial.FF
 Only in (first) C:\chad_anl -> solve_lset_reinitial.orig
 Only in (first) C:\chad_anl -> solve_lset_reinitial.TZ
 Only in (first) C:\chad_anl ->
 solve_lset_reinitial_module_c.FF
 Only in (first) C:\chad_anl ->
 solve_lset_reinitial_module_c.orig
 Only in (first) C:\chad_anl ->
 solve_lset_reinitial_module_c.TZ
 Only in (second) C:\chad_uiuc -> solve_mat.FF
 Only in (second) C:\chad_uiuc -> solve_or_res_nl_cont.FF
 Only in (second) C:\chad_uiuc -> solve_or_res_nl_tmp.FF
 Only in (second) C:\chad_uiuc -> solve_or_res_nl_uvw.FF

```

Older      file      ->      C:\chad_anl\solve_ort_s.FF      *than*
C:\chad_uiuc\solve_ort_s.FF
Only in (second) C:\chad_uiuc -> solve_ort_t5.FF
Older      file      ->      C:\chad_anl\solve_ort_v.FF      *than*
C:\chad_uiuc\solve_ort_v.FF
Older      file      ->      C:\chad_anl\solve_ort_v6.FF      *than*
C:\chad_uiuc\solve_ort_v6.FF
Only in (second) C:\chad_uiuc -> solve_petsc_cont.FF
Only in (second) C:\chad_uiuc -> solve_petsc_cont_par.FF
Only in (second) C:\chad_uiuc -> solve_petsc_cont_par_2.FF
Older      file      ->      C:\chad_anl\solve_spc.FF      *than*
C:\chad_uiuc\solve_spc.FF
Older      file      ->      C:\chad_anl\solve_spc_module_c.FF  *than*
C:\chad_uiuc\solve_spc_module_c.FF
Older      file      ->      C:\chad_anl\solve_strs.FF      *than*
C:\chad_uiuc\solve_strs.FF
Only in (second) C:\chad_uiuc -> solve_strs_module_c.FF
Only in (second) C:\chad_uiuc -> solve_tmf.FF
Only in (second) C:\chad_uiuc -> solve_tmf_module_c.FF
Only in (first) C:\chad_anl -> solve_tmp.FF
Only in (first) C:\chad_anl -> solve_tmp.orig
Older      file      ->      C:\chad_anl\solve_tmp_module_c.FF  *than*
C:\chad_uiuc\solve_tmp_module_c.FF
Only in (first) C:\chad_anl -> solve_uvw.FF
Older      file      ->      C:\chad_anl\solve_uvw_lag.FF      *than*
C:\chad_uiuc\solve_uvw_lag.FF
Older      file      ->      C:\chad_anl\solve_uvw_module_c.FF  *than*
C:\chad_uiuc\solve_uvw_module_c.FF
Older      file      ->      C:\chad_anl\solve_xyz.FF      *than*
C:\chad_uiuc\solve_xyz.FF
Only in (second) C:\chad_uiuc -> solve_xyz_avg.FF
Only in (second) C:\chad_uiuc -> solve_xyz_avg_module_c.FF
Only in (second) C:\chad_uiuc -> solve_xyz_elp.FF
Only in (second) C:\chad_uiuc -> solve_xyz_elp_module_c.FF
Only in (second) C:\chad_uiuc -> solve_xyz_elp_sor.FF
Older      file      ->      C:\chad_anl\solve_xyz_module_c.FF  *than*
C:\chad_uiuc\solve_xyz_module_c.FF
Only in (second) C:\chad_uiuc -> solve_xyz_rot.FF
Only in (second) C:\chad_uiuc -> solve_xyz_rot_module_c.FF
Only in (second) C:\chad_uiuc -> solve_xyz_smth.FF
Only in (second) C:\chad_uiuc -> solve_xyz_smth_module_c.FF
Only in (second) C:\chad_uiuc -> solve_xyz_spc.FF
Only in (second) C:\chad_uiuc -> solve_xyz_spc_module_c.FF
Older      file      ->      C:\chad_anl\sort.FF      *than*
C:\chad_uiuc\sort.FF
Older      file      ->      C:\chad_anl\species.FF      *than*
C:\chad_uiuc\species.FF

```

Older file -> C:\chad_anl\species_module_c.FF *than*
 C:\chad_uiuc\species_module_c.FF
 Only in (second) C:\chad_uiuc -> spray_module_d.FF
 Only in (second) C:\chad_uiuc -> srcschem_ener.FF
 Only in (second) C:\chad_uiuc -> srcschem_spcmass.FF
 Older file -> C:\chad_anl\srcsexp.FF *than*
 C:\chad_uiuc\srcsexp.FF
 Only in (second) C:\chad_uiuc -> srcsexp_update_driver.FF
 Older file -> C:\chad_anl\srcsexp_update_module_d.FF *than*
 C:\chad_uiuc\srcsexp_update_module_d.FF
 Older file -> C:\chad_anl\summary.FF *than*
 C:\chad_uiuc\summary.FF
 Only in (second) C:\chad_uiuc -> symmetrize_module_d.FF
 Older file -> C:\chad_anl\timedata_module_c.FF *than*
 C:\chad_uiuc\timedata_module_c.FF
 Older file -> C:\chad_anl\timerdata_module_c.FF *than*
 C:\chad_uiuc\timerdata_module_c.FF
 Older file -> C:\chad_anl\timers_module_d.FF *than*
 C:\chad_uiuc\timers_module_d.FF
 Older file -> C:\chad_anl\timstp.FF *than*
 C:\chad_uiuc\timstp.FF
 Older file -> C:\chad_anl\timstpr.FF *than*
 C:\chad_uiuc\timstpr.FF
 Older file -> C:\chad_anl\trans_coef_module_d.FF *than*
 C:\chad_uiuc\trans_coef_module_d.FF
 Only in (second) C:\chad_uiuc -> transient.FF
 Only in (second) C:\chad_uiuc -> units_module_c.FF
 Only in (second) C:\chad_uiuc -> update_connssgn.FF
 Older file -> C:\chad_anl\User_bc_flags_module_d.FF *than*
 C:\chad_uiuc\User_bc_flags_module_d.FF
 Older file -> C:\chad_anl\User_bcs.FF *than*
 C:\chad_uiuc\User_bcs.FF
 Older file -> C:\chad_anl\User_chem.FF *than*
 C:\chad_uiuc\User_chem.FF
 Older file -> C:\chad_anl\User_in.FF *than*
 C:\chad_uiuc\User_in.FF
 Only in (second) C:\chad_uiuc -> User_intrfc_nullify.FF
 Only in (first) C:\chad_anl -> User_lset_initial.FF
 Only in (second) C:\chad_uiuc -> User_monitor.FF
 Older file -> C:\chad_anl\User_movmsh.FF *than*
 C:\chad_uiuc\User_movmsh.FF
 Older file -> C:\chad_anl\User_out.FF *than*
 C:\chad_uiuc\User_out.FF
 Older file -> C:\chad_anl\User_prep.FF *than*
 C:\chad_uiuc\User_prep.FF
 Older file -> C:\chad_anl\User_pst.FF *than*
 C:\chad_uiuc\User_pst.FF

Older file -> C:\chad_anl\User_srcs.FF *than*
 C:\chad_uiuc\User_srcs.FF
 Older file -> C:\chad_anl\User_srcsexp.FF *than*
 C:\chad_uiuc\User_srcsexp.FF
 Only in (second) C:\chad_uiuc -> User_test.FF
 Older file -> C:\chad_anl\User_wall.FF *than*
 C:\chad_uiuc\User_wall.FF
 Only in (first) C:\chad_anl -> User_wall.orig
 Older file -> C:\chad_anl\version.FF *than*
 C:\chad_uiuc\version.FF
 Older file -> C:\chad_anl\volume_module_d.FF *than*
 C:\chad_uiuc\volume_module_d.FF
 Older file -> C:\chad_anl\vorticity.FF *than*
 C:\chad_uiuc\vorticity.FF
 Only in (second) C:\chad_uiuc -> work_bcs.FF
 Only in (second) C:\chad_uiuc -> write_arrays.FF
 Only in (second) C:\chad_uiuc -> write_ctrl_setup.FF
 Older file -> C:\chad_anl\write_dmp.FF *than*
 C:\chad_uiuc\write_dmp.FF
 Only in (second) C:\chad_uiuc -> write_dmp_ctrl.FF
 Only in (second) C:\chad_uiuc -> write_max.FF
 Older file -> C:\chad_anl\write_mesh.FF *than*
 C:\chad_uiuc\write_mesh.FF
 Only in (second) C:\chad_uiuc -> write_out_ctrl.FF
 Older file -> C:\chad_anl\write_pst.FF *than*
 C:\chad_uiuc\write_pst.FF
 Older file -> C:\chad_anl\write_pst_avs.FF *than*
 C:\chad_uiuc\write_pst_avs.FF
 Older file -> C:\chad_anl\write_pst_ens.FF *than*
 C:\chad_uiuc\write_pst_ens.FF
 Older file -> C:\chad_anl\write_pst_fv.FF *than*
 C:\chad_uiuc\write_pst_fv.FF
 Older file -> C:\chad_anl\write_pst_gmv.FF *than*
 C:\chad_uiuc\write_pst_gmv.FF
 Only in (second) C:\chad_uiuc -> write_residuals.FF
 Older file -> C:\chad_anl\write_tty.FF *than*
 C:\chad_uiuc\write_tty.FF
 Only in (second) C:\chad_uiuc -> zero_cavity_tildes.FF
 Only in (second) C:\chad_uiuc -> zero_wall_tildes.FF
 Only in (second) C:\chad_uiuc -> zero_walls.FF

F.2 Diff of CHAD 5.0 to CHAD_UIUC

```
Only in (second) C:\chad_uiuc -> anl_data_module_c.FF
Older file -> C:\chad_5.0\anti_diff_mats.FF *than*
C:\chad_uiuc\anti_diff_mats.FF
Older file -> C:\chad_5.0\api.FF *than* C:\chad_uiuc\api.FF
Older file -> C:\chad_5.0\api_module_c.FF *than*
C:\chad_uiuc\api_module_c.FF
Older file -> C:\chad_5.0\arrays_glo_module_c.FF *than*
C:\chad_uiuc\arrays_glo_module_c.FF
Older file -> C:\chad_5.0\assoc_flow.FF *than*
C:\chad_uiuc\assoc_flow.FF
Older file -> C:\chad_5.0\bcd_data_module_c.FF *than*
C:\chad_uiuc\bcd_data_module_c.FF
Older file -> C:\chad_5.0\bcs.FF *than* C:\chad_uiuc\bcs.FF
Older file -> C:\chad_5.0\bcs_dep_module_d.FF *than*
C:\chad_uiuc\bcs_dep_module_d.FF
Older file -> C:\chad_5.0\burn_fracs_prog.FF *than*
C:\chad_uiuc\burn_fracs_prog.FF
Older file -> C:\chad_5.0\burn_fracs_react.FF *than*
C:\chad_uiuc\burn_fracs_react.FF
Older file -> C:\chad_5.0\burn_intervals.FF *than*
C:\chad_uiuc\burn_intervals.FF
Older file -> C:\chad_5.0\cellface_module_d.FF *than*
C:\chad_uiuc\cellface_module_d.FF
Older file -> C:\chad_5.0\chad_main.FF *than*
C:\chad_uiuc\chad_main.FF
Older file -> C:\chad_5.0\CHAD_P.FF *than*
C:\chad_uiuc\CHAD_P.FF
Older file -> C:\chad_5.0\com_setup.FF *than*
C:\chad_uiuc\com_setup.FF
Older file -> C:\chad_5.0\conserve.FF *than*
C:\chad_uiuc\conserve.FF
Older file -> C:\chad_5.0\conserve_mats.FF *than*
C:\chad_uiuc\conserve_mats.FF
Older file -> C:\chad_5.0\constants_module_c.FF *than*
C:\chad_uiuc\constants_module_c.FF
Older file -> C:\chad_5.0\construct_eletype.FF *than*
C:\chad_uiuc\construct_eletype.FF
Older file -> C:\chad_5.0\cycle_loop.FF *than*
C:\chad_uiuc\cycle_loop.FF
Older file -> C:\chad_5.0\dante_int_module_d.FF *than*
C:\chad_uiuc\dante_int_module_d.FF
Only in (second) C:\chad_uiuc -> debug-print.FF
Only in (first) C:\chad_5.0 -> DEFINE.HH
```

```

Older      file      ->      C:\chad_5.0\deltas.FF      *than*
C:\chad_uiuc\deltas.FF
Older file -> C:\chad_5.0\dimensions_module_c.FF *than*
C:\chad_uiuc\dimensions_module_c.FF
Older file -> C:\chad_5.0\dissip_n_module_d.FF *than*
C:\chad_uiuc\dissip_n_module_d.FF
Older file -> C:\chad_5.0\distribute_mesh_module_d.FF
*than* C:\chad_uiuc\distribute_mesh_module_d.FF
Older      file      ->      C:\chad_5.0\driver.FF      *than*
C:\chad_uiuc\driver.FF
Older      file      ->      C:\chad_5.0\eigen.FF      *than*
C:\chad_uiuc\eigen.FF
Older file -> C:\chad_5.0\element_module_c.FF *than*
C:\chad_uiuc\element_module_c.FF
Older      file      ->      C:\chad_5.0\enthdiff.FF      *than*
C:\chad_uiuc\enthdiff.FF
Older      file      ->      C:\chad_5.0\enthflux.FF      *than*
C:\chad_uiuc\enthflux.FF
Older      file      ->      C:\chad_5.0\eos_module_d.FF      *than*
C:\chad_uiuc\eos_module_d.FF
Older file -> C:\chad_5.0\eos_ses_module_d.FF *than*
C:\chad_uiuc\eos_ses_module_d.FF
Older file -> C:\chad_5.0\eosdata_module_c.FF *than*
C:\chad_uiuc\eosdata_module_c.FF
Older      file      ->      C:\chad_5.0\eospac_tables.FF      *than*
C:\chad_uiuc\eospac_tables.FF
Only in (first) C:\chad_5.0 -> eospac_types.HH
Older file -> C:\chad_5.0\error_check_module_d.FF *than*
C:\chad_uiuc\error_check_module_d.FF
Older      file      ->      C:\chad_5.0\flow_stress.FF      *than*
C:\chad_uiuc\flow_stress.FF
Older      file      ->      C:\chad_5.0\flux_corr.FF      *than*
C:\chad_uiuc\flux_corr.FF
Older      file      ->      C:\chad_5.0\gather_module_d.FF      *than*
C:\chad_uiuc\gather_module_d.FF
Older      file      ->      C:\chad_5.0\grads_module_d.FF      *than*
C:\chad_uiuc\grads_module_d.FF
Older      file      ->      C:\chad_5.0\hydro.FF      *than*
C:\chad_uiuc\hydro.FF
Older      file      ->      C:\chad_5.0\hydro_exp.FF      *than*
C:\chad_uiuc\hydro_exp.FF
Older      file      ->      C:\chad_5.0\hydro_module_c.FF      *than*
C:\chad_uiuc\hydro_module_c.FF
Older      file      ->      C:\chad_5.0\ieflux.FF      *than*
C:\chad_uiuc\ieflux.FF
Older      file      ->      C:\chad_5.0\improv_solt.FF      *than*
C:\chad_uiuc\improv_solt.FF

```

```

Older      file      ->      C:\chad_5.0\improve_sol.FF      *than*
C:\chad_uiuc\improve_sol.FF
Older      file      ->      C:\chad_5.0\init_flags.FF      *than*
C:\chad_uiuc\init_flags.FF
Older      file      ->      C:\chad_5.0\init_mesh.FF      *than*
C:\chad_uiuc\init_mesh.FF
Older      file      ->      C:\chad_5.0\inout_module_d.FF    *than*
C:\chad_uiuc\inout_module_d.FF
Older      file      ->      C:\chad_5.0\inoutdata_module_c.FF *than*
C:\chad_uiuc\inoutdata_module_c.FF
Older      file      ->      C:\chad_5.0\intrfc_init.FF      *than*
C:\chad_uiuc\intrfc_init.FF
Older      file      ->      C:\chad_5.0\intrfc_module_d.FF  *than*
C:\chad_uiuc\intrfc_module_d.FF
Older      file      ->      C:\chad_5.0\intrfc_nullify.FF   *than*
C:\chad_uiuc\intrfc_nullify.FF
Older      file      ->      C:\chad_5.0\iterdata_module_c.FF *than*
C:\chad_uiuc\iterdata_module_c.FF
Older      file      ->      C:\chad_5.0\ke_consts_module_c.FF *than*
C:\chad_uiuc\ke_consts_module_c.FF
Older      file      ->      C:\chad_5.0\kepsilon.FF        *than*
C:\chad_uiuc\kepsilon.FF
Older      file      ->      C:\chad_5.0\kepsilon_module_c.FF *than*
C:\chad_uiuc\kepsilon_module_c.FF
Older      file      ->      C:\chad_5.0\Kinds_module_c.FF   *than*
C:\chad_uiuc\Kinds_module_c.FF
Only in (first) C:\chad_5.0 -> lchad
Only in (first) C:\chad_5.0 -> macros.HH
Only in (second) C:\chad_uiuc -> matrix_coeffs.FF
Older      file      ->      C:\chad_5.0\mesh_connectivity.FF *than*
C:\chad_uiuc\mesh_connectivity.FF
Older      file      ->      C:\chad_5.0\mesh_conns.FF      *than*
C:\chad_uiuc\mesh_conns.FF
Older      file      ->      C:\chad_5.0\mesh_conns_par.FF   *than*
C:\chad_uiuc\mesh_conns_par.FF
Older      file      ->      C:\chad_5.0\mesh_consistency.FF *than*
C:\chad_uiuc\mesh_consistency.FF
Older      file      ->      C:\chad_5.0\mesh_fix.FF        *than*
C:\chad_uiuc\mesh_fix.FF
Older      file      ->      C:\chad_5.0\mesh_move.FF       *than*
C:\chad_uiuc\mesh_move.FF
Older      file      ->      C:\chad_5.0\mesh_move_lag_module_d.FF *than*
C:\chad_uiuc\mesh_move_lag_module_d.FF
Older      file      ->      C:\chad_5.0\mesh_update.FF      *than*
C:\chad_uiuc\mesh_update.FF
Older      file      ->      C:\chad_5.0\mesh_vel_only.FF    *than*
C:\chad_uiuc\mesh_vel_only.FF

```

```

Older      file      ->      C:\chad_5.0\misc_update.FF      *than*
C:\chad_uiuc\misc_update.FF
Older      file      ->      C:\chad_5.0\ncoupler.FF      *than*
C:\chad_uiuc\ncoupler.FF
Older      file      ->      C:\chad_5.0\nodetypes_module_c.FF      *than*
C:\chad_uiuc\nodetypes_module_c.FF
Older      file      ->      C:\chad_5.0\options_module_c.FF      *than*
C:\chad_uiuc\options_module_c.FF
Older      file      ->      C:\chad_5.0\orthdx_s.FF      *than*
C:\chad_uiuc\orthdx_s.FF
Older      file      ->      C:\chad_5.0\orthdx_v.FF      *than*
C:\chad_uiuc\orthdx_v.FF
Older      file      ->      C:\chad_5.0\orthdx_v5.FF      *than*
C:\chad_uiuc\orthdx_v5.FF
Older      file      ->      C:\chad_5.0\orthdx_v6.FF      *than*
C:\chad_uiuc\orthdx_v6.FF
Only in (second) C:\chad_uiuc -> parallel_debug.FF
Older      file      ->      C:\chad_5.0\Parallel_module_c.FF      *than*
C:\chad_uiuc\Parallel_module_c.FF
Older      file      ->      C:\chad_5.0\part_mesh_module_d.FF      *than*
C:\chad_uiuc\part_mesh_module_d.FF
Only in (second) C:\chad_uiuc -> petsc_chad_finalize.FF
Only in (second) C:\chad_uiuc -> petsc_chad_init.FF
Older      file      ->      C:\chad_5.0\physdata_module_c.FF      *than*
C:\chad_uiuc\physdata_module_c.FF
Older      file      ->      C:\chad_5.0\physics.FF      *than*
C:\chad_uiuc\physics.FF
Older      file      ->      C:\chad_5.0\physics_module_c.FF      *than*
C:\chad_uiuc\physics_module_c.FF
Older      file      ->      C:\chad_5.0\piston.FF      *than*
C:\chad_uiuc\piston.FF
Older      file      ->      C:\chad_5.0\pistondata_module_c.FF      *than*
C:\chad_uiuc\pistondata_module_c.FF
Older      file      ->      C:\chad_5.0\polar_decomp.FF      *than*
C:\chad_uiuc\polar_decomp.FF
Older      file      ->      C:\chad_5.0\precond_module_d.FF      *than*
C:\chad_uiuc\precond_module_d.FF
Older      file      ->      C:\chad_5.0\prep.FF      *than*
C:\chad_uiuc\prep.FF
Older      file      ->      C:\chad_5.0\read_arrays.FF      *than*
C:\chad_uiuc\read_arrays.FF
Older      file      ->      C:\chad_5.0\read_input.FF      *than*
C:\chad_uiuc\read_input.FF
Older      file      ->      C:\chad_5.0\read_mesh.FF      *than*
C:\chad_uiuc\read_mesh.FF
Older      file      ->      C:\chad_5.0\read_msh_or_rst_dist.FF      *than*
C:\chad_uiuc\read_msh_or_rst_dist.FF

```

Older file -> C:\chad_5.0\read_msh_or_rst_iope.FF *than*
 C:\chad_uiuc\read_msh_or_rst_iope.FF
 Older file -> C:\chad_5.0\read_rst.FF *than*
 C:\chad_uiuc\read_rst.FF
 Older file -> C:\chad_5.0\read_spc.FF *than*
 C:\chad_uiuc\read_spc.FF
 Older file -> C:\chad_5.0\readvars_module_d.FF *than*
 C:\chad_uiuc\readvars_module_d.FF
 Older file -> C:\chad_5.0\res_lin_cont.FF *than*
 C:\chad_uiuc\res_lin_cont.FF
 Older file -> C:\chad_5.0\res_lin_hydro.FF *than*
 C:\chad_uiuc\res_lin_hydro.FF
 Older file -> C:\chad_5.0\res_lin_ke.FF *than*
 C:\chad_uiuc\res_lin_ke.FF
 Older file -> C:\chad_5.0\res_lin_mat.FF *than*
 C:\chad_uiuc\res_lin_mat.FF
 Older file -> C:\chad_5.0\res_lin_spc.FF *than*
 C:\chad_uiuc\res_lin_spc.FF
 Older file -> C:\chad_5.0\res_lin_strs.FF *than*
 C:\chad_uiuc\res_lin_strs.FF
 Older file -> C:\chad_5.0\res_lin_tmf.FF *than*
 C:\chad_uiuc\res_lin_tmf.FF
 Older file -> C:\chad_5.0\res_lin_tmp.FF *than*
 C:\chad_uiuc\res_lin_tmp.FF
 Older file -> C:\chad_5.0\res_lin_tmp_exp.FF *than*
 C:\chad_uiuc\res_lin_tmp_exp.FF
 Older file -> C:\chad_5.0\res_lin_uvw.FF *than*
 C:\chad_uiuc\res_lin_uvw.FF
 Older file -> C:\chad_5.0\res_lin_uvw_exp.FF *than*
 C:\chad_uiuc\res_lin_uvw_exp.FF
 Older file -> C:\chad_5.0\res_lin_uvw_lag.FF *than*
 C:\chad_uiuc\res_lin_uvw_lag.FF
 Older file -> C:\chad_5.0\res_lin_xyz.FF *than*
 C:\chad_uiuc\res_lin_xyz.FF
 Older file -> C:\chad_5.0\res_lin_xyz_avg.FF *than*
 C:\chad_uiuc\res_lin_xyz_avg.FF
 Older file -> C:\chad_5.0\res_lin_xyz_elp.FF *than*
 C:\chad_uiuc\res_lin_xyz_elp.FF
 Older file -> C:\chad_5.0\res_lin_xyz_elp_sor.FF *than*
 C:\chad_uiuc\res_lin_xyz_elp_sor.FF
 Older file -> C:\chad_5.0\res_lin_xyz_rot.FF *than*
 C:\chad_uiuc\res_lin_xyz_rot.FF
 Older file -> C:\chad_5.0\res_lin_xyz_spc.FF *than*
 C:\chad_uiuc\res_lin_xyz_spc.FF
 Older file -> C:\chad_5.0\run_statistics.FF *than*
 C:\chad_uiuc\run_statistics.FF

Older file -> C:\chad_5.0\scatter_module_d.FF *than*
 C:\chad_uiuc\scatter_module_d.FF
 Only in (first) C:\chad_5.0 -> schad
 Older file -> C:\chad_5.0\ser_ary_transfer.FF *than*
 C:\chad_uiuc\ser_ary_transfer.FF
 Older file -> C:\chad_5.0\set_units.FF *than*
 C:\chad_uiuc\set_units.FF
 Older file -> C:\chad_5.0\shuffle_module_d.FF *than*
 C:\chad_uiuc\shuffle_module_d.FF
 Older file -> C:\chad_5.0\solve_cg_v.FF *than*
 C:\chad_uiuc\solve_cg_v.FF
 Older file -> C:\chad_5.0\solve_cont_module_c.FF *than*
 C:\chad_uiuc\solve_cont_module_c.FF
 Older file -> C:\chad_5.0\solve_gcr_s.FF *than*
 C:\chad_uiuc\solve_gcr_s.FF
 Older file -> C:\chad_5.0\solve_gcr_t5.FF *than*
 C:\chad_uiuc\solve_gcr_t5.FF
 Older file -> C:\chad_5.0\solve_gcr_v.FF *than*
 C:\chad_uiuc\solve_gcr_v.FF
 Older file -> C:\chad_5.0\solve_gcr_v6.FF *than*
 C:\chad_uiuc\solve_gcr_v6.FF
 Older file -> C:\chad_5.0\solve_gcr_v_precond.FF *than*
 C:\chad_uiuc\solve_gcr_v_precond.FF
 Older file -> C:\chad_5.0\solve_gmr_s.FF *than*
 C:\chad_uiuc\solve_gmr_s.FF
 Older file -> C:\chad_5.0\solve_gmr_t5.FF *than*
 C:\chad_uiuc\solve_gmr_t5.FF
 Older file -> C:\chad_5.0\solve_gmr_v.FF *than*
 C:\chad_uiuc\solve_gmr_v.FF
 Older file -> C:\chad_5.0\solve_gmr_v6.FF *than*
 C:\chad_uiuc\solve_gmr_v6.FF
 Older file -> C:\chad_5.0\solve_gmr_v_precond.FF *than*
 C:\chad_uiuc\solve_gmr_v_precond.FF
 Older file -> C:\chad_5.0\solve_hydro.FF *than*
 C:\chad_uiuc\solve_hydro.FF
 Older file -> C:\chad_5.0\solve_hydro_exp.FF *than*
 C:\chad_uiuc\solve_hydro_exp.FF
 Older file -> C:\chad_5.0\solve_hydro_module_c.FF *than*
 C:\chad_uiuc\solve_hydro_module_c.FF
 Older file -> C:\chad_5.0\solve_ke.FF *than*
 C:\chad_uiuc\solve_ke.FF
 Older file -> C:\chad_5.0\solve_ke_module_c.FF *than*
 C:\chad_uiuc\solve_ke_module_c.FF
 Older file -> C:\chad_5.0\solve_mat.FF *than*
 C:\chad_uiuc\solve_mat.FF
 Older file -> C:\chad_5.0\solve_or_res_nl_cont.FF *than*
 C:\chad_uiuc\solve_or_res_nl_cont.FF

Older file -> C:\chad_5.0\solve_or_res_nl_tmp.FF *than*
 C:\chad_uiuc\solve_or_res_nl_tmp.FF
 Older file -> C:\chad_5.0\solve_or_res_nl_uvw.FF *than*
 C:\chad_uiuc\solve_or_res_nl_uvw.FF
 Older file -> C:\chad_5.0\solve_ort_s.FF *than*
 C:\chad_uiuc\solve_ort_s.FF
 Older file -> C:\chad_5.0\solve_ort_t5.FF *than*
 C:\chad_uiuc\solve_ort_t5.FF
 Older file -> C:\chad_5.0\solve_ort_v.FF *than*
 C:\chad_uiuc\solve_ort_v.FF
 Older file -> C:\chad_5.0\solve_ort_v6.FF *than*
 C:\chad_uiuc\solve_ort_v6.FF
 Only in (second) C:\chad_uiuc -> solve_petsc_cont.FF
 Only in (second) C:\chad_uiuc -> solve_petsc_cont_par.FF
 Only in (second) C:\chad_uiuc -> solve_petsc_cont_par_2.FF
 Older file -> C:\chad_5.0\solve_spc.FF *than*
 C:\chad_uiuc\solve_spc.FF
 Older file -> C:\chad_5.0\solve_spc_module_c.FF *than*
 C:\chad_uiuc\solve_spc_module_c.FF
 Older file -> C:\chad_5.0\solve_strs.FF *than*
 C:\chad_uiuc\solve_strs.FF
 Older file -> C:\chad_5.0\solve_strs_module_c.FF *than*
 C:\chad_uiuc\solve_strs_module_c.FF
 Older file -> C:\chad_5.0\solve_tmf.FF *than*
 C:\chad_uiuc\solve_tmf.FF
 Older file -> C:\chad_5.0\solve_tmf_module_c.FF *than*
 C:\chad_uiuc\solve_tmf_module_c.FF
 Older file -> C:\chad_5.0\solve_tmp_module_c.FF *than*
 C:\chad_uiuc\solve_tmp_module_c.FF
 Older file -> C:\chad_5.0\solve_uvw_lag.FF *than*
 C:\chad_uiuc\solve_uvw_lag.FF
 Older file -> C:\chad_5.0\solve_uvw_module_c.FF *than*
 C:\chad_uiuc\solve_uvw_module_c.FF
 Older file -> C:\chad_5.0\solve_xyz.FF *than*
 C:\chad_uiuc\solve_xyz.FF
 Older file -> C:\chad_5.0\solve_xyz_avg.FF *than*
 C:\chad_uiuc\solve_xyz_avg.FF
 Older file -> C:\chad_5.0\solve_xyz_avg_module_c.FF *than*
 C:\chad_uiuc\solve_xyz_avg_module_c.FF
 Older file -> C:\chad_5.0\solve_xyz_elp.FF *than*
 C:\chad_uiuc\solve_xyz_elp.FF
 Older file -> C:\chad_5.0\solve_xyz_elp_module_c.FF *than*
 C:\chad_uiuc\solve_xyz_elp_module_c.FF
 Older file -> C:\chad_5.0\solve_xyz_elp_sor.FF *than*
 C:\chad_uiuc\solve_xyz_elp_sor.FF
 Older file -> C:\chad_5.0\solve_xyz_module_c.FF *than*
 C:\chad_uiuc\solve_xyz_module_c.FF


```

Older file -> C:\chad_5.0\solve_xyz_rot.FF *than*
C:\chad_uiuc\solve_xyz_rot.FF
Older file -> C:\chad_5.0\solve_xyz_rot_module_c.FF *than*
C:\chad_uiuc\solve_xyz_rot_module_c.FF
Older file -> C:\chad_5.0\solve_xyz_smth.FF *than*
C:\chad_uiuc\solve_xyz_smth.FF
Older file -> C:\chad_5.0\solve_xyz_smth_module_c.FF *than*
C:\chad_uiuc\solve_xyz_smth_module_c.FF
Older file -> C:\chad_5.0\solve_xyz_spc.FF *than*
C:\chad_uiuc\solve_xyz_spc.FF
Older file -> C:\chad_5.0\solve_xyz_spc_module_c.FF *than*
C:\chad_uiuc\solve_xyz_spc_module_c.FF
Older file -> C:\chad_5.0\sort.FF *than*
C:\chad_uiuc\sort.FF
Older file -> C:\chad_5.0\species.FF *than*
C:\chad_uiuc\species.FF
Older file -> C:\chad_5.0\species_module_c.FF *than*
C:\chad_uiuc\species_module_c.FF
Older file -> C:\chad_5.0\spray_module_d.FF *than*
C:\chad_uiuc\spray_module_d.FF
Older file -> C:\chad_5.0\srcschem_ener.FF *than*
C:\chad_uiuc\srcschem_ener.FF
Older file -> C:\chad_5.0\srcschem_spcmass.FF *than*
C:\chad_uiuc\srcschem_spcmass.FF
Older file -> C:\chad_5.0\srcsexp.FF *than*
C:\chad_uiuc\srcsexp.FF
Older file -> C:\chad_5.0\srcsexp_update_driver.FF *than*
C:\chad_uiuc\srcsexp_update_driver.FF
Older file -> C:\chad_5.0\srcsexp_update_module_d.FF *than*
C:\chad_uiuc\srcsexp_update_module_d.FF
Older file -> C:\chad_5.0\summary.FF *than*
C:\chad_uiuc\summary.FF
Older file -> C:\chad_5.0\symmetrize_module_d.FF *than*
C:\chad_uiuc\symmetrize_module_d.FF
Older file -> C:\chad_5.0\timedata_module_c.FF *than*
C:\chad_uiuc\timedata_module_c.FF
Older file -> C:\chad_5.0\timerdata_module_c.FF *than*
C:\chad_uiuc\timerdata_module_c.FF
Older file -> C:\chad_5.0\timers_module_d.FF *than*
C:\chad_uiuc\timers_module_d.FF
Older file -> C:\chad_5.0\timstp.FF *than*
C:\chad_uiuc\timstp.FF
Older file -> C:\chad_5.0\timstpr.FF *than*
C:\chad_uiuc\timstpr.FF
Older file -> C:\chad_5.0\trans_coef_module_d.FF *than*
C:\chad_uiuc\trans_coef_module_d.FF

```

```

Older      file      ->      C:\chad_5.0\transient.FF      *than*
C:\chad_uiuc\transient.FF
Older      file      ->      C:\chad_5.0\units_module_c.FF  *than*
C:\chad_uiuc\units_module_c.FF
Older      file      ->      C:\chad_5.0\update_connssgn.FF  *than*
C:\chad_uiuc\update_connssgn.FF
Older      file      ->      C:\chad_5.0\User_bc_flags_module_d.FF *than*
C:\chad_uiuc\User_bc_flags_module_d.FF
Older      file      ->      C:\chad_5.0\User_bcs.FF      *than*
C:\chad_uiuc\User_bcs.FF
Older      file      ->      C:\chad_5.0\User_chem.FF      *than*
C:\chad_uiuc\User_chem.FF
Older      file      ->      C:\chad_5.0\User_in.FF       *than*
C:\chad_uiuc\User_in.FF
Older      file      ->      C:\chad_5.0\User_intrfc_nullify.FF *than*
C:\chad_uiuc\User_intrfc_nullify.FF
Only in (second) C:\chad_uiuc -> User_monitor.FF
Older      file      ->      C:\chad_5.0\User_movmsh.FF    *than*
C:\chad_uiuc\User_movmsh.FF
Older      file      ->      C:\chad_5.0\User_out.FF      *than*
C:\chad_uiuc\User_out.FF
Older      file      ->      C:\chad_5.0\User_prep.FF     *than*
C:\chad_uiuc\User_prep.FF
Older      file      ->      C:\chad_5.0\User_pst.FF      *than*
C:\chad_uiuc\User_pst.FF
Older      file      ->      C:\chad_5.0\User_srcs.FF     *than*
C:\chad_uiuc\User_srcs.FF
Older      file      ->      C:\chad_5.0\User_srcsexp.FF  *than*
C:\chad_uiuc\User_srcsexp.FF
Older      file      ->      C:\chad_5.0\User_test.FF     *than*
C:\chad_uiuc\User_test.FF
Older      file      ->      C:\chad_5.0\User_wall.FF     *than*
C:\chad_uiuc\User_wall.FF
Older      file      ->      C:\chad_5.0\version.FF      *than*
C:\chad_uiuc\version.FF
Older      file      ->      C:\chad_5.0\volume_module_d.FF *than*
C:\chad_uiuc\volume_module_d.FF
Older      file      ->      C:\chad_5.0\vorticity.FF     *than*
C:\chad_uiuc\vorticity.FF
Only in (second) C:\chad_uiuc -> work_bcs.FF
Older      file      ->      C:\chad_5.0\write_arrays.FF  *than*
C:\chad_uiuc\write_arrays.FF
Older      file      ->      C:\chad_5.0\write_ctrl_setup.FF *than*
C:\chad_uiuc\write_ctrl_setup.FF
Older      file      ->      C:\chad_5.0\write_dmp.FF     *than*
C:\chad_uiuc\write_dmp.FF

```

```

Older   file    ->    C:\chad_5.0\write_dmp_ctrl.FF      *than*
C:\chad_uiuc\write_dmp_ctrl.FF
Only in (second) C:\chad_uiuc -> write_max.FF
Older   file    ->    C:\chad_5.0\write_mesh.FF         *than*
C:\chad_uiuc\write_mesh.FF
Older   file    ->    C:\chad_5.0\write_out_ctrl.FF      *than*
C:\chad_uiuc\write_out_ctrl.FF
Older   file    ->    C:\chad_5.0\write_pst.FF           *than*
C:\chad_uiuc\write_pst.FF
Older   file    ->    C:\chad_5.0\write_pst_avs.FF       *than*
C:\chad_uiuc\write_pst_avs.FF
Older   file    ->    C:\chad_5.0\write_pst_ens.FF       *than*
C:\chad_uiuc\write_pst_ens.FF
Older   file    ->    C:\chad_5.0\write_pst_fv.FF       *than*
C:\chad_uiuc\write_pst_fv.FF
Older   file    ->    C:\chad_5.0\write_pst_gmv.FF       *than*
C:\chad_uiuc\write_pst_gmv.FF
Only in (second) C:\chad_uiuc -> write_residuals.FF
Older   file    ->    C:\chad_5.0\write_tty.FF          *than*
C:\chad_uiuc\write_tty.FF
Only in (second) C:\chad_uiuc -> zero_cavity_tildes.FF
Only in (second) C:\chad_uiuc -> zero_wall_tildes.FF
Only in (second) C:\chad_uiuc -> zero_walls.FF

```

F.3 Modified and New Source Routines

ANL DATA MODULE C

This module contains constants and flags used in the ANL and UIUC modifications.

```
31c31
< C      gravx      = gravity acceleration in x direction (negative)
---
> c      gravx      = gravity acceleration in x direction (negative)
34a35
> C      vexpansion = coefficient of volumetric expansion
51c52,53
<
---
> c      matrix_osolv= logical flag. If true, solves continuity equation
without ctz reformulation, but with PETSc ! DTR
> c      alt_bc      = logical flag. If true, uses alternate formulation
for boundary conditions ! DTR
108a111
> c
177a181
> c      i_no_e      = flag to turn off solution of energy equation
208c212
< c      imatrixbug =
---
> c      imatrixbug = spit out debugging info for matrix solving
217a222
> c      ictz        = use Tzanos approach for tilde velocities
218a224,282
> c
> c      Define a few things for DTR specifics ! DTR
> c
> c      imon_node_u = nodes for convergence monitoring
> c      imon_node_v = they are designed for single processor usage
> c      imon_node_w =
> c      imon_node_t =
> c      imon_node_p =
> c      imon_node_k =
> c      imon_node_e =
>
> c      plot_res    = flag to plot global residuals for convergence
> c                  monitoring
> c      plot_mon    = flag to plot variable monitors for convergence
> c                  monitoring
>
> c      norm_fact_u = normalization factor for u-mom residuals
> c      norm_fact_v = normalization factor for v-mom residuals
> c      norm_fact_w = normalization factor for w-mom residuals
> c      norm_fact_t = normalization factor for temp residuals
> c      norm_fact_p = normalization factor for pressure residuals
```

```

>
> c      r_red_fact_u = residual reduction factor for u-mom
> c      r_red_fact_v = residual reduction factor for v-mom
> c      r_red_fact_w = residual reduction factor for w-mom
> c      r_red_fact_t = residual reduction factor for t
> c      r_red_fact_p = residual reduction factor for p
>
> c      conv_res_u   = convergence threshold for u-momentum residuals
> c      conv_res_v   = convergence threshold for v-momentum residuals
> c      conv_res_w   = convergence threshold for w-momentum residuals
> c      conv_res_t   = convergence threshold for temperature residuals
> c      conv_res_p   = convergence threshold for pressure residuals
>
> c      slv_tol_s= factor by which to adjust the scalar solver
tolerances
> c      slv_tol_v= factor by which to adjust the vectorized solver
tolerances
>
> c      mon_x      = x-coordinate for convergence monitoring
> c      mon_y      = designed for parallel usage
> c      mon_z
>
> c      float_rtol = flag to allow petsc relative tolerance limits to
> c                  dynamically change -- experimental
>
> c      petsc_rtol  = dynamically changing petsc relative tolerance
when
> c                  the float_rtol flag is being used
> c
> c      urf_m       = under-relaxation factor for momentum
> c      urf_p       = under-relaxation factor for pressure
> c      urf_t       = under-relaxation factor for temperature
> c      use_urfp    = flag turning on urf for pressure
> c      use_urfm    = flag turning on urf for momentum
> c      use_urft    = flag turning on urf for temperature
> c      porous_media= flag for using porous media approximation
> c      imodel_id   = an integer flag for defining models used in User
> c                  routines without having to comment and uncomment
> c                  constantly.
> c -----
---
>
227,229c291,293
<      &          ,outavs
<      &          ,outfv
<      &          ,outgm
---
> c      &          ,outavs
> c      &          ,outfv
> c      &          ,outgm
231a296,304
>      &          ,plot_res    ! DTR
>      &          ,plot_mon    ! DTR
>      &          ,float_rtol   ! DTR
>      &          ,porous_media ! DTR
>      &          ,use_urft     ! DTR
>      &          ,use_urfp     ! DTR

```

```

>      &      ,use_urfm      ! DTR
>      &      ,use_urftke    ! DTR
>      &      ,use_urfeps    ! DTR
272d344
<
286a359
>      &      ,i_no_e
295a369,377
>      &      ,ictz      !DTR
>      &      ,imon_node_u  !DTR
>      &      ,imon_node_v  !DTR
>      &      ,imon_node_w  !DTR
>      &      ,imon_node_t  !DTR
>      &      ,imon_node_p  !DTR
>      &      ,imon_node_k  !DTR
>      &      ,imon_node_e  !DTR
>      &      ,imodel_id
319c401,426
<
---
>      &      ,mon_x      !DTR
>      &      ,mon_y      !DTR
>      &      ,mon_z      !DTR
>      &      ,norm_fact_u  !DTR
>      &      ,norm_fact_v  !DTR
>      &      ,norm_fact_w  !DTR
>      &      ,norm_fact_t  !DTR
>      &      ,norm_fact_p  !DTR
>      &      ,r_red_fact_u  !DTR
>      &      ,r_red_fact_v  !DTR
>      &      ,r_red_fact_w  !DTR
>      &      ,r_red_fact_t  !DTR
>      &      ,r_red_fact_p  !DTR
>      &      ,conv_res_u    !DTR
>      &      ,conv_res_v    !DTR
>      &      ,conv_res_w    !DTR
>      &      ,conv_res_t    !DTR
>      &      ,conv_res_p    !DTR
>      &      ,slv_tol_s     !DTR
>      &      ,slv_tol_v     !DTR
>      &      ,petsc_rtol !DTR
>      &      ,urf_p        !DTR
>      &      ,urf_m        !DTR
>      &      ,urf_t        !DTR
>      &      ,urf_tke      !DTR
>      &      ,urf_eps      !DTR

323c430
<      &      ::  rho11 , rho12 , etalset, sigma ,      ! CTZ
---
>      &      ::  rho11 , rho12 , etalset, sigma ,      ! CTZ
325c432
<      &      mull1 , mul2      ! CTZ
---
>      &      mull1 , mul2, vexansion      ! CTZ-DTR
352c459,460
<      &      ,matrix_solv      ! CTZ

```

```

---
>      &      ,matrix_solv, matrix_osolv      !   CTZ / DTR
>      &      ,alt_bc ! DTR

```

ARRAYS_GLO_MODULE_C

This routine contains modifications for the ANL first order upwinding scheme, and some experimental boundary condition arrays developed at UIUC.

```

656a657
>
664a666
>
751a754,755
>      &      NBC_INFLOW ,NBC_OUTFLOW , ! DTR
>      &      O_RESCL      ,O_RESC      , ! DTR
767c771,772
<
---
>      &      , VOLFLOLDM      ! CTZ
>      &      , WALL_BDRY ! DTR
1269a1275,1276
>      &      NBC_INFLOW ,NBC_OUTFLOW , ! DTR
>      &      O_RESCL      ,O_RESC      , ! DTR
1383a1391,1392
>      &      , VOLFLOLDM      ! CTZ
>      &      , WALL_BDRY      ! DTR
1393c1402,1403
<
---
> CHPFD_DISTRIBUTE VOLFLOLDM (      _BLK_)      ! CTZ
> CHPFD_DISTRIBUTE WALL_BDRY (      _BLK_)      ! DTR
2009a2020,2023
>      &      NBC_INFLOW (  nnodemax1), ! DTR
>      &      NBC_OUTFLOW (  nnodemax1), ! DTR
>      &      O_RESCL      (  nnodemax1), ! DTR
>      &      O_RESC      (  nnodemax1), ! DTR
2073a2088,2089
>      &      VOLFLOLDM (  nconns1 ),      ! CTZ
>      &      WALL_BDRY (  nconns1 ),      ! DTR
4052a4069,4072
>      &      NBC_INFLOW , ! DTR
>      &      NBC_OUTFLOW , ! DTR
>      &      O_RESCL      , ! DTR
>      &      O_RESC      , ! DTR
4116a4137,4138
>      &      VOLFLOLDM      ,      ! CTZ
>      &      WALL_BDRY      ,      ! DTR

```

BCS

Defines terms used in the reformulated continuity equation and matrix based solutions.

```

351a352
>
353,354c354,355
<      use      Parallel_module_c      ,only:
<      &      global_any
---
>      use      Parallel_module_c !      ,only:
> c      &      global_any, global_sum
368c369
<      &      bc_uvw
---
>      &      bc_uvw, bc_uvw_dtr ! DTR
371a373,374
>      use      anl_data_module_c      ,only:
>      &      ictz, alt_bc
373d375
<
418c420
<      &      :: isp
---
>      &      :: isp, ii !cdtr
508a511,512
>      real      (kind      =real_kind  )
>      &      rinlet_sum, tmp
626c630,631
<      & MFRACINFLOW,MFRACOUTFLOW)
---
>      & MFRACINFLOW,MFRACOUTFLOW,
>      & P_OPEN)      !CDTR
629d633
<
774a779,786
>      if (alt_bc) then
>
>      call bc_uvw_dtr('change',USOURCE,VSOURCE,WSOURCE,
>      &      UWALL,VWALL,WWALL,
>      &      NXINFLOW,NYINFLOW,NZINFLOW)
>
>      else
>
782a795,796
>      end if
>
801d814
<
888d900
<
892c904
<
---
>
907c919
<
---
>

```



```

911a924
>
914a928,937
>
> CTZ    Compute momentum sources due to inflow at the boundaries. This
> CTZ    is used in the computation of interface fluxes in
cellface_tilde.
>
> c      if ((ictz.eq.1).or.(matrix_osolv))then
>         SRCMOMU=SRCMASS*UENTER
>         SRCMOMV=SRCMASS*VENTER
>         SRCMOMW=SRCMASS*WENTER
> c      end if
>
970c993,998
<      WHERE(OUTFLOW_TYPE.EQ.1
---
> CDTR   CTZ defines this in User_bcs
> CDTR   Note, it would be overwritten anyway as User_bcs call is later
> CDTR   in this routine.
>
>       if(ictz.ne.1) then !CDTR
>         WHERE(OUTFLOW_TYPE.EQ.1
980,981c1008,1010
<         P_OPEN=POUTFLOW
<         ENDWHERE
---
>         P_OPEN=POUTFLOW
>         ENDWHERE
>       end if
1091a1121,1124
> c      call debug_print(1)
>
> c      stop
>
1096a1130
>

```

BCS_DEP_MODULE_D

Defines terms used in the reformulated continuity equation and matrix based solutions.
Contains some modifications for an experimental alternate boundary condition.

```

23a24
>
26c27,28
<      &          CONTAB
---
>      &          CONTAB
>      &          ,NBC_INFLOW, NBC_OUTFLOW ! DTR
42c44
<      public  :: bc_s      ,bc_spc      ,bc_uvw      ,bc_xyz
---

```

```
>      public  :: bc_s ,bc_spc ,bc_uvw ,bc_xyz,  bc_uvw_dtr
562a565,783
```

CELLFACE_MODULE_D

Primarily calculates cell face velocity terms for the reformulated continuity equation.

```
41a42
>
46a48,49
>      use      anl_data_module_c      ,only:
>      &        iup_rho, iup_uvw, iup_tmp,ictz
47a51,52
>      use      hydro_module_c      ,only:
>      &        VFXUPW      ! DTR-CTZ
63c68,69
<      public  :: cellface_hyd      ,
---
>      public  :: cellface_coeff      , ! DTR
>      &        cellface_hyd      ,
68c74
<      &        cellface_tilde_change      ,
---
>      &        cellface_tilde_change      , ! DTR-CTZ
87a94,95
>
>
93c101
< C      Interfaces.
---
> C      wInterfaces.
97a106
>      module procedure cellface_tilde_ctz
104a114,115
>      module procedure cellface_tilde_change_ctz_mom
>      module procedure cellface_tilde_change_ctz_cont
111a123
>
390c402,404
<      &      DISTXM,DISTYM,DISTZM,HM)
---
>      &      DISTXM,DISTYM,DISTZM,
>      &      VFXUPW, !DTR
>      &      HM)
520a535
>      &      ,VFXUPW ! CTZ
522c537
<
---
> CHPFD_DIDDISTRIBUTE VFXUPW
569a585,593
> C      If first order upwind (iup_tmp =1) then compute HM as below
> C
>      if(iup_tmp.eq.1) then      ! CTZ
>      VFXUPW =1.0
>      WHERE(COURANTM.LT.0.0)
```

```

>          VFXUPW = -1.0
>          ENDWHERE
>
>          HM = .5*(HT(1,:)*(1.+ VFXUPW) + HT(2,:)*(1.-VFXUPW))
570a595
>          else      ! CTZ
711a737,739
>          end if ! CTZ upwind
>
>
*****
967a996
>          &          VFXUPW          ,
1027a1057
>          &          VFXUPW,
1039a1070
>          &          VFXUPW,          !ctz-dtr
1056a1088
>          &          VFXUPW,          !ctz-dtr
1072c1104,1106
<          &          DISTXM,DISTYM,DISTZM,HM)
---
>          &          DISTXM,DISTYM,DISTZM,
>          &          VFXUPW, !CTZ
>          &          HM)
1666a1701
>          &          VFXUPW,          !ctz-dtr
1766a1802
>          &          ,NODEST
1798a1835,1838
>          real          (kind          =real_kind ),
>          &          dimension ( nconns )
>          &          :: VFXUPW          ! CTZ
>          CHPDFD_DISTRIBUTE VFXUPW          (          _BLK_)          ! CTZ
1833a1874,1876
>          integer          (kind          =int_kind )
>          &          :: ii,k1,k2
>
1972a2016,2025
>          if(ictz.eq.1)then
>          WHERE(VOLFLOWM.GE.zero)      ! CTZ
>          VFXUPW = one
>          ENDWHERE
>
>          WHERE(VOLFLOWM.LT.zero)      ! CTZ
>          VFXUPW = -one
>          ENDWHERE
>          end if
>
2023a2077
>          &          VFXUPW,
2070a2125,2127
>          C          VFXUPW          = indicator of median-mesh flux direction
! CTZ
>          C          computed on time level n values of COURANTM
>          C
2091a2149,2150

```

```

>          use          anl_data_module_c          ,only:
>          &          ictz
2134a2194
> CHPFD_DISTRIBUTE VFXUPW          (          _BLK_)
2146a2207,2210
>          real          (kind          =real_kind  ),
>          &          dimension          (          nconns  )
>          &          :: VFXUPW          ! CTZ
> CHPFD_DISTRIBUTE VFXUPW          (          _BLK_)          ! CTZ
2182a2247,2255
>          if(iup_rho.eq.1) then          ! CTZ
>              VFXUPW =one
>              WHERE(COURANTM.LT.0.0)
>                  VFXUPW = -one
>              ENDWHERE
>
>              RHOM = half*(RHOT(1,:)*(one+ VFXUPW)+RHOT(2,:)*(one-
VFXUPW))
>
>          else          ! CTZ
2296a2370,2372
>          endif          !          upwind          CTZ
>
>
3085a3162,4345
>          subroutine cellface_tilde_ctz(
>          &          PT,UT,VT,WT,
>          &          UTILDEM,VTILDEM,WTILDEM,VOLFLOWM)
>
>
#####
>
> C          Purpose:
>
> C          Computes the "tilde" velocity and volumetric flow rate
> C          at median-mesh boundaries.
>
> C          Input Variables:
>
> C          PN          = Current pressure at the nodes.
> C          UN          = Current x-direction velocity at the nodes.
> C          VN          = Current y-direction velocity at the nodes.
> C          WN          = Current z-direction velocity at the nodes.
> C          PT          = Current value of pressure gathered at
> C                      connection terminuses.
> C          UT          = Current x-direction velocity gathered at
> C                      connection terminuses.
> C          VT          = Current y-direction velocity gathered at
> C                      connection terminuses.
> C          WT          = Current z-direction velocity gathered at
> C                      connection terminuses.
> C          DPKDXN          = X-component of  $P+(2/3)*\rho*K$  gradient at
the
> C                      nodes.
> C          DPKDYN          = Y-component of  $P+(2/3)*\rho*K$  gradient at
the
> C                      nodes.

```

```

> C          DPKDZN          = Z-component of  $P+(2/3)*\rho*K$  gradient at
the
> C          nodes.
> C          POLDN          = Time-level n pressure at the nodes.
> C          RHOOLDN        = Time-level n value of density at the nodes.
> C          POLDT          = Time-level n value of pressure gathered at
> C          connection terminuses.
> C          RHOOLDT        = Time-level n value of density gathered at
> C          connection terminuses.
> C          UOLDT          = Time-level n x-direction velocity gathered
at
> C          connection terminuses.
> C          VOLDT          = Time-level n y-direction velocity gathered
at
> C          connection terminuses.
> C          WOLDT          = Time-level n z-direction velocity gathered
at
> C          connection terminuses.
>
> C          FLAGSTRSBC     = A logical flag that is set to .true.
> C                          where we want S.A to be zero while
computing
> C                          the divergence of the stress deviator, and
is
> C                          set to .false. elsewhere.
> C          STRSXX         = XX-component of material-stress tensor.
> C          STRSXY         = XY-component of material-stress tensor.
> C          STRSXZ         = XZ-component of material-stress tensor.
> C          STRSYY         = YY-component of material-stress tensor.
> C          STRSYZ         = YZ-component of material-stress tensor.
> C          STRSZZ         = ZZ-component of material-stress tensor.
>
> C          Output Variables:
>
> C          UTILDEM        = X-direction "tilde" velocity at median-mesh
> C          boundary.
> C          VTILDEM        = Y-direction "tilde" velocity at median-mesh
> C          boundary.
> C          WTILDEM        = Z-direction "tilde" velocity at median-mesh
> C          boundary.
> C          VOLFLOWM        = Volumetric flow rate crossing the median-
mesh
> C          boundary.
>
>
>
C
#####
>
> C          Module list and other preliminaries
>
>          use          arrays_glo_module_c          , only:
>          &          NODETYPE          ,
>          &          X,Y,Z,          !DTR
>          &          STRSXX          , STRSXY          , STRSXZ          ,
>          &          STRSYY          , STRSYZ          , STRSZZ          ,
>          &          DELTAL          , UMESH          , VMESH          , WMESH
,
>          &          AXM          , AYM          , AZM          ,

```

```

>      &          VOLSWPTRATEM,
>      &          RHO , VOL , MASSOLD , UOLD , VOLD , WOLD,
! CTZ
>      &          AXN_OPEN , AYN_OPEN , AZN_OPEN , U , V , W,
! CTZ
>      &          UWALL, VWALL, WWALL, P,
>      &          NODEST ! DTR-debug
>      use      eosdata_module_c ,only:
>      &          mat_stress
>      use      hydro_module_c ,only:
>      &          FLAGSTRSBC, NOFLUXM, flagnofluxm
>      &          , RHOM, RHOT ! CTZ / DTR
>      &          , VFXUPW ! CTZ / DTR
>      &          , CNUM , CNVM , CNWM ! CTZ / DTR
>      &          , CNEIUM , CNEIVM , CNEIWM ! CTZ / DTR
>      &          , AMUL, AMUT, AMULM, AMUTM
>      use      nodetypes_module_c ,only:
>      &          ntp_intrfc, ntp_free_src, ntp_inflow
>      use      options_module_c ,only:
>      &          flowtype
>      use      physdata_module_c ,only:
>      &          ftlag, geometry,pgs ! /geometry DTR
>      use      physics_module_c ,only:
>      &          USOURCE , VSOURCE, WSOURCE,
>      &          SRCMOMU , SRCMOMV , SRCMOMW ,
>      &          P_OPEN, ! CTZ
>      &          COURANTM !DTR
>      use      timedata_module_c ,only:
>      &          dt
>      use      gather_module_d ,only:
>      &          gathert_sn ,gathert_vn
>      use      scatter_module_d ,only:
>      &          scattert_sn ,scattert_vn
>      use      grads_module_d ,only:
>      &          divn_strsn, gradt_sm,gradt_sn, difft_tau
>
>      implicit none
>
>
##### C
>
> C Other global and local variables.
>
>
##### C
>
> C Global variables being passed through the argument list.
>
>      real (kind =real_kind ),
>      &      dimension ( 2,nconns )
> c      &      intent (in ) !CDTR
>      &      :: PT ,UT ,VT ,WT
>
> CHPDFD_DISTRIBUTE PT (_STR_,_BLK_)
> CHPDFD_DISTRIBUTE UT (_STR_,_BLK_)
> CHPDFD_DISTRIBUTE VT (_STR_,_BLK_)
> CHPDFD_DISTRIBUTE WT (_STR_,_BLK_)

```

```

>
>
>      real (kind =real_kind ), ! CTZ
>      &      dimension ( 2,nconns )
>      &      :: CNUT , CNVT , CNWT ,
>      &      CNEIUT , CNEIVT , CNEIWT
> CHPFD_DISTRIBUTE CNUT      (_STR_,_BLK_)
> CHPFD_DISTRIBUTE CNVT      (_STR_,_BLK_)
> CHPFD_DISTRIBUTE CNWT      (_STR_,_BLK_)
> CHPFD_DISTRIBUTE CNEIUT    (_STR_,_BLK_)
> CHPFD_DISTRIBUTE CNEIVT    (_STR_,_BLK_)
> CHPFD_DISTRIBUTE CNEIWT    (_STR_,_BLK_)
>
>      real      (kind      =real_kind ),
>      &      dimension (  nconns ),
>      &      intent      (out      )
>      &      :: UTILDEM      ,VTILDEM      ,WTILDEM      ,VOLFLOWM
> CHPFD_DISTRIBUTE UTILDEM    (      _BLK_ )
> CHPFD_DISTRIBUTE VTILDEM    (      _BLK_ )
> CHPFD_DISTRIBUTE WTILDEM    (      _BLK_ )
> CHPFD_DISTRIBUTE VOLFLOWM   (      _BLK_ )
>
>      real      (kind      =real_kind ),
>      &      dimension (  nnodemax)
>      &      ::CNU   ,   CNV   ,   CNW   ,
>      &      CNEIU ,   CNEIV ,   CNEIW
> CHPFD_DISTRIBUTE CNU      (      _BLK_ )      ! DTR
> CHPFD_DISTRIBUTE CNV      (      _BLK_ )
> CHPFD_DISTRIBUTE CNW      (      _BLK_ )
> CHPFD_DISTRIBUTE CNEIU    (      _BLK_ )      ! DTR
> CHPFD_DISTRIBUTE CNEIV    (      _BLK_ )
> CHPFD_DISTRIBUTE CNEIW    (      _BLK_ )
>
>
>

```

C

```

>
> C      Local variables.
>      integer      (kind      =int_kind )
>      &      ::ii,k1,k2,jj !dtr
>
>      real      (kind      =real_kind )
>      &      :: velmax,rpgs2
>
>      real      (kind      =real_kind ),
>      &      dimension (  nnodemax)
>      &      :: DUDTN      ,DVDTN      ,DWDTN      ,
>      &      SCRATCHN1      ,
>      &      DIVTAUXN      ,DIVTAUYN      ,DIVTAUZN
>      &      ,DPDXN      ,DPDYN      ,DPDZN      !DTR
> CHPFD_DISTRIBUTE DUDTN      (      _BLK_ )
> CHPFD_DISTRIBUTE DVDTN      (      _BLK_ )
> CHPFD_DISTRIBUTE DWDTN      (      _BLK_ )
> CHPFD_DISTRIBUTE SCRATCHN1  (      _BLK_ )
> CHPFD_DISTRIBUTE DPDXN      (      _BLK_ )
> CHPFD_DISTRIBUTE DPDYN      (      _BLK_ )
> CHPFD_DISTRIBUTE DPDZN      (      _BLK_ )

```



```

>          ENDWHERE
>
>          WHERE (VOLFLOWM.LT.0.0)          ! CTZ
>              VFXUPW = -one          ! CTZ
>          ENDWHERE
>
>          call gathert_sn(RHO, RHOT)
>
>          if(iup_rho.eq.1) then
>              RHOM=half*(RHOT(1,:)*(1.+VFXUPW)+RHOT(2,:)*(1.-VFXUPW))
>          else
>              RHOM=half*(RHOT(1,:)+RHOT(2,:))
>          endif
>
>
>
> C          Compute coefficient CNU (coefficient of node velocity)
> C          !!! VOLFLOWM may not be known !!!
> CDTR      Certainly unknown during a restart at this time
>
> CDTR      2nd term equation (11) Tzanos notes
>
>          CNUT(1,:) = half*(1.+ VFXUPW(:))*RHOM(:)*VOLFLOWM(:)
>          CNUT(2,:) = -half*(1.- VFXUPW(:))*RHOM(:)*VOLFLOWM(:)
>          CNVT(1,:) = CNUT(1,:)
>          CNVT(2,:) = CNUT(2,:)
>          CNWT(1,:) = CNUT(1,:)
>          CNWT(2,:) = CNUT(2,:)
>
>
> C          Compute the sum of the products Cneighbor*Uneighbor
>
>          CNEIUT(1,:) = CNUT(2,:)*UT(2,:)
>          CNEIUT(2,:) = CNUT(1,:)*UT(1,:)
>          CNEIVT(1,:) = CNVT(2,:)*VT(2,:)
>          CNEIVT(2,:) = CNVT(1,:)*VT(1,:)
>          CNEIWT(1,:) = CNWT(2,:)*WT(2,:)
>          CNEIWT(2,:) = CNWT(1,:)*WT(1,:)
>
>
> C          Accumulate from connections to nodes to compute CNU
>
>          call scattert_vn('add', CNU , CNV, CNW,
> &          CNUT, CNVT, CNWT )
>
>
> C          Compute volumetric flow and donor density at outflow
> C          boundaries. The volumetric flow, QFLOWOUT, and donor density,
> C          RHOOUTFLOW, are defined where the flow is actually out of the
> C          problem (QFLOWOUT always negative), and are set to zero at all
> C          non-outflow or inflow/outflow boundaries where the flow is
> C          into the problem.
>
>          SCRATCHN1 = zero
>          SCRATCHN1= (U-UMESH)*AXN_OPEN
> &          + (V-VMESH)*AYN_OPEN
> &          + (W-WMESH)*AZN_OPEN
>
>          WHERE (P_OPEN .GE.hugenum .OR.
> C          &          SCRATCHN1.GE.zero .OR.          ! CTZ
> C          &          NODETYPE .EQ.ntp_free .OR.          ! CTZ

```

```

>      &      NODETYPE .EQ.ntp_free_src.OR.
>      &      NODETYPE .EQ.ntp_inflow. OR.      ! CTZ
>      &      NODETYPE .EQ.ntp_intrfc
>      &      )
>      SCRATCHN1 = zero
>      ELSEWHERE
>      SCRATCHN1 = RHO * SCRATCHN1
>      ENDWHERE
>
> CDTR      Equation (11) Tzanos notes
>
>      CNU = VOL*RHO/dt - SCRATCHN1 + CNU
>      CNV = VOL*RHO/dt - SCRATCHN1 + CNV
>      CNW = VOL*RHO/dt - SCRATCHN1 + CNW
>
> C      Compute the sum of the products Cneighbor*Uneighbor
> c      scatter from terminus to nodes
>
>      call scattert_vn('add', CNEIU , CNEIV, CNEIW,
>      &      CNEIUT, CNEIVT, CNEIWT )
>
> C      Add to CNEIU the term VOL*RHOLD*UOLD/dt and dived by CNU
>
>
> CDTR      This is equation (16) in Tzanos notes
>
>      call difft_tau(UT,VT,WT,
>      & AMUL,AMUT,AMULM,AMUTM,
>      & TAUDOTAXM=DUDTM, TAUDOTAYM=DVDTM, TAUDOTAZM=DWDTM,
>      & DIVTAUXN=DIVTAUXN, DIVTAUYN=DIVTAUYN, DIVTAUZN=DIVTAUZN)
>
>      CNEIU = half*(CNEIU + MASSOLD*UOLD/dt+
>      &      VOL*DIVTAUXN + SRCMOMU)/CNU
>      CNEIV = half*(CNEIV + MASSOLD*VOLD/dt+
>      &      VOL*DIVTAUYN + SRCMOMV)/CNV
>      CNEIW = half*(CNEIW + MASSOLD*WOLD/dt+
>      &      VOL*DIVTAUZN + SRCMOMW)/CNW
>
>
> CDTR      Equation (17) nodebased, in Tzanos' notes
>
>      CNU = half*VOL/CNU
>      CNV = half*VOL/CNV
>      CNW = half*VOL/CNW
>
> C      To Compute averages transfer from nodes to connection terminuses
>
>      call gathert_vn(CNU, CNV, CNW,
>      &      CNUT, CNVT, CNWT)
>
>      call gathert_vn(CNEIU, CNEIV, CNEIW,
>      &      CNEIUT, CNEIVT, CNEIWT)
>
> C      Compute averages at connections
>
> CDTR      This is RHS of (17) in Tzanos
>      CNUM = CNUT(1,:) + CNUT(2,:)

```



```

>
> C          Computes the change in the "tilde" velocity
> C          at median-mesh boundaries for the CTZ formulation.
>
> C          Input Variables:
>
> C          UT          = Current x-direction velocity change
gathered at
> C          connection terminuses.
> C          VT          = Current y-direction velocity change
gathered at
> C          connection terminuses.
> C          WT          = Current z-direction velocity change
gathered at
> C          connection terminuses.
>
> C          FLAGSTRSBC = A logical flag that is set to .true.
> C                      where we want S.A to be zero while
computing
> C                      the divergence of the stress deviator, and
is
> C                      set to .false. elsewhere.
> C          Output Variables:
>
> C          UTILDEM      = X-direction "tilde" velocity change at
median-mesh
> C          boundary.
> C          VTILDEM      = Y-direction "tilde" velocity change at
median-mesh
> C          boundary.
> C          WTILDEM      = Z-direction "tilde" velocity change at
median-mesh
> C          boundary.
>
>
>
##### C
#####
>
> C          Module list and other preliminaries
>
>          use          arrays_glo_module_c          ,only:
>          &          NODETYPE      ,X,Y,Z,P,
>          &          UMESH          ,VMESH          ,WMESH          ,
>          &          AXM            ,AYM            ,AZM            ,
>          &          VOLSWPTRATEM, UWALL, VWALL, WWALL
>          &          , VOL , MASSOLD , UOLD , VOLD , WOLD      ! CTZ
> C          &          , RHO , VOL , MASSOLD , UOLD , VOLD , WOLD
! CTZ
>          &          , AXN_OPEN , AYN_OPEN , AZN_OPEN , U , V ,
W,NODEST ! CTZ
>          use          hydro_module_c          ,only:
>          &          AMUL , AMUT , AMULM, AMUTM, RHOT, NOFLUXM,      !
CTZ
>          &          VFXUPW,flagnofluxm,CNUM,CNVM,CNWM          ! DTR
>          use          nodetypes_module_c          ,only:
>          &          ntp_inflow
>          use          options_module_c          ,only:
>          &          flowtype

```

```

>      use      physics_module_c      ,only:
>      &        P_OPEN, SRCMOMU, SRCMOMV, SRCMOMW
>      use      physdata_module_c      ,only:
>      &        ftlag
>      &        ,pgs      !DTR
>      use      physics_module_c      ,only:
>      &        USOURCE, VSOURCE, WSOURCE
>      &        ,SRCMOMU , SRCMOMV , SRCMOMW , P_OPEN      !CTZ
>      use      timedata_module_c      ,only:
>      &        dt
>      &        , ncyc      !
CTZ
>      use      gather_module_d      ,only:
>      &        gathert_sn ,gathert_vn
>      use      scatter_module_d      ,only:
>      &        scattert_vn      ! CTZ
>      use      grads_module_d      ,only:
>      &        divn_strsn
>      &        ,gradt_sm , diff_t_tau,gradt_sn
>      use      nodetypes_module_c      ! CTZ
>
>      implicit none
>
>
>
#####
>
> C      Other global and local variables.
>
>
>
C

```

```

>
> C      Global variables being passed through the argument list.
>
>      real      (kind      =real_kind ),      ! CTZ
>      &          dimension (  nnodemax)
>      &          :: UN , VN , WN , RHO      ! CTZ
> CHPFD_DISTRIBUTE UN      (      _BLK_)
> CHPFD_DISTRIBUTE VN      (      _BLK_)
> CHPFD_DISTRIBUTE WN      (      _BLK_)
> CHPFD_DISTRIBUTE RHO      (      _BLK_)
>
>      real      (kind      =real_kind ),
>      &          dimension (  2,nconns ),
>      &          intent (in )
>      &          :: UUT      ,VVT      ,WWT
> CHPFD_DISTRIBUTE UUT      (_STR_,_BLK_)
> CHPFD_DISTRIBUTE VVT      (_STR_,_BLK_)
> CHPFD_DISTRIBUTE WWT      (_STR_,_BLK_)
>
>      real      (kind      =real_kind ),
>      &          dimension (  nconns ),
>      &          intent (out )
>      &          :: UUTILDEM ,VVTILDEM ,WWTILDEM
> CHPFD_DISTRIBUTE UUTILDEM (      _BLK_)
> CHPFD_DISTRIBUTE VVTILDEM (      _BLK_)
> CHPFD_DISTRIBUTE WWTILDEM (      _BLK_)
>

```

```

>      real      (kind      =real_kind ),
>      &          dimension  (   nconns  ),
>      &          intent    (in         )
>      &          :: VOLFLOWM
> CHPFD_DISTRIBUTE VOLFLOWM      (      _BLK_)
>
>      real      (kind      =real_kind ),      ! CTZ
>      &          dimension  (   nconns  )
>      &          :: RHOM
> CHPFD_DISTRIBUTE RHOM          (      _BLK_)      ! CTZ
>
>
>
> C      Local variables.
>
>      real      (kind      =real_kind )
>      &          :: velmax
>
>      real      (kind      =real_kind ),
>      &          dimension  (   nnodemax)
>      &          :: SCRATCHN1 ,
>      &          DIVTAUXN    ,DIVTAUYN    ,DIVTAUZN
> CHPFD_DISTRIBUTE SCRATCHN1 (      _BLK_)
> CHPFD_DISTRIBUTE DIVTAUXN  (      _BLK_)
> CHPFD_DISTRIBUTE DIVTAUYN  (      _BLK_)
> CHPFD_DISTRIBUTE DIVTAUZN  (      _BLK_)
>
>      real      (kind      =real_kind ),
>      &          dimension  (   nconns  )
>      &          :: DELTAUTM ,DELTAVTM    ,DELTAWTM
> CHPFD_DISTRIBUTE DELTAUTM  (      _BLK_)
> CHPFD_DISTRIBUTE DELTAVTM  (      _BLK_)
> CHPFD_DISTRIBUTE DELTAWTM  (      _BLK_)
> CHPFD_DISTRIBUTE SCRATCHM1 (      _BLK_)
>
>      real      (kind      =real_kind ),      ! CTZ
>      &          dimension  (   nnodemax)
>      &          :: CNU      , CNV      , CNW      ,      ! CTZ
>      &          CNEIU      , CNEIV      , CNEIW      ! CTZ
> CHPFD_DISTRIBUTE CNU        (      _BLK_)
> CHPFD_DISTRIBUTE CNV        (      _BLK_)
> CHPFD_DISTRIBUTE CNW        (      _BLK_)
> CHPFD_DISTRIBUTE CNEIU      (      _BLK_)
> CHPFD_DISTRIBUTE CNEIV      (      _BLK_)
> CHPFD_DISTRIBUTE CNEIW      (      _BLK_)
>
>      real      (kind      =real_kind ),      ! CTZ
>      &          dimension  ( 2,nconns )
>      &          :: CNUT , CNVT , CNWT ,
>      &          CNEIUT , CNEIVT , CNEIWT
> CHPFD_DISTRIBUTE CNUT        (_STR_,_BLK_)
> CHPFD_DISTRIBUTE CNVT        (_STR_,_BLK_)
> CHPFD_DISTRIBUTE CNWT        (_STR_,_BLK_)
> CHPFD_DISTRIBUTE CNEIUT      (_STR_,_BLK_)
> CHPFD_DISTRIBUTE CNEIVT      (_STR_,_BLK_)
> CHPFD_DISTRIBUTE CNEIWT      (_STR_,_BLK_)

```

C

[illegible]

```

>         CNEIWT(2,:) = CNWT(1,:)*WWT(1,:)
>
> C         Accumulate from connections to nodes to compute CNU
>
>         call scattert_vn('add', CNU , CNV, CNW,
> &         CNUT, CNVT, CNWT )
>
> C         Compute contribution from outflow boundaries. Note that
> C         current CHAD version uses first order upwinding at
> C         outflow(see solve_cont)
>
> C         Compute volumetric flow and donor density at outflow
> C         boundaries. The volumetric flow, QFLOWOUT, and donor density,
> C         RHOOUTFLOW, are defined where the flow is actually out of the
> C         problem (QFLOWOUT always negative), and are set to zero at all
> C         non-outflow or inflow/outflow boundaries where the flow is
> C         into the problem.
>
> C         SCRATCHN1 is used below with a negative sign because
> C         AXN_OPEN etc are negative
> C         d(rho*u)/dt = momentum(in) -momentum(out)+...
>
>
>         SCRATCHN1 = zero
>         SCRATCHN1= (UN-UMESH)*AXN_OPEN
> &                 + (VN-VMESH)*AYN_OPEN
> &                 + (WN-WMESH)*AZN_OPEN
>         WHERE (P_OPEN .GE.hugenum .OR.
> C &         SCRATCHN1.GE.zero .OR.           ! CTZ
> C &         NODETYPE .EQ.ntp_free .OR.
> &         NODETYPE .EQ.ntp_free_src.OR.
> &         NODETYPE .EQ.ntp_inflow. OR.       ! CTZ
> &         NODETYPE .EQ.ntp_intrfc
> &         )
>         SCRATCHN1 = zero
>         ELSEWHERE
>         SCRATCHN1 = RHO * SCRATCHN1
>         ENDWHERE
>
>         CNU = VOL*RHO/dt - SCRATCHN1 + CNU
>         CNV = VOL*RHO/dt - SCRATCHN1 + CNV
>         CNW = VOL*RHO/dt - SCRATCHN1 + CNW
>
> C         Accumulate from connections to nodes to compute CNEIU:
> C         (sum of Cneighbor*Uneighbor)
>
>         call scattert_vn('add', CNEIU , CNEIV, CNEIW,
> &         CNEIUT, CNEIVT, CNEIWT )
>
>         CNEIU = half*CNEIU/CNU
>         CNEIV = half*CNEIV/CNV
>         CNEIW = half*CNEIW/CNW
>
> C         To Compute averages transfer from nodes to connection terminuses
>
>         call gathert_vn(CNEIU, CNEIV, CNEIW, CNEIUT, CNEIVT, CNEIWT)
>

```



```

> C                                boundary.
> C                                = Volumetric flow rate crossing the median-
mesh
> C                                boundary.
>
>
#####
>
> C      Module list and other preliminaries
>
>      use      arrays_glo_module_c      ,only:
>      &        NODETYPE      ,X,Y,Z,P,
>      &        STRSXX      ,STRSXY      ,STRSXZ      ,
>      &        STRSYY      ,STRSYZ      ,STRSZZ      ,
>      &        DELTAL      ,UMESH      ,VMESH      ,WMESH
,
>      &        AXM      ,AYM      ,AZM      ,
>      &        VOLSWPTRATEM, UWALL, VWALL, WWALL
>      &        , VOL , MASSOLD , UOLD , VOLD , WOLD      ! CTZ
> C      &        , RHO , VOL , MASSOLD , UOLD , VOLD , WOLD
! CTZ
>      &        , AXN_OPEN , AYN_OPEN , AZN_OPEN , U , V ,
W,NODEST ! CTZ
>      use      hydro_module_c      ,only:
>      &        AMUL , AMUT , AMULM, AMUTM, RHOT, NOFLUXM,      !
CTZ
>      &        VFXUPW,flagnofluxm,CNUM,CNVM,CNWM      ! DTR
>      use      nodetypes_module_c      ,only:
>      &        ntp_inflow
>      use      options_module_c      ,only:
>      &        flowtype
>      use      physics_module_c      ,only:
>      &        P_OPEN, SRCMOMU, SRCMOMV, SRCMOMW
>      use      physdata_module_c      ,only:
>      &        ftlag
>      &        ,pgs      !DTR
>      use      physics_module_c      ,only:
>      &        USOURCE, VSOURCE, WSOURCE
>      &        ,SRCMOMU , SRCMOMV , SRCMOMW , P_OPEN      !CTZ
>      use      timedata_module_c      ,only:
>      &        dt
>      &        , ncyc      !
CTZ
>      use      gather_module_d      ,only:
>      &        gathert_sn ,gathert_vn
>      use      scatter_module_d      ,only:
>      &        scattert_vn      ! CTZ
>      use      grads_module_d      ,only:
>      &        divn_strsn
>      &        ,gradt_sm , diff_t_tau,gradt_sn
>      use      nodetypes_module_c      ! CTZ
>
>      implicit none
>
>
#####
>

```

```

> C      Other global and local variables.
>
>


---


>
> C      Global variables being passed through the argument list.
>
>
>      real      (kind      =real_kind ),      ! CTZ
>      &          dimension  (  nnodemax)
>      &          :: UN , VN , WN , RHO      ! CTZ
> CHPFD_DISTRIBUTE UN      (      _BLK_)
> CHPFD_DISTRIBUTE VN      (      _BLK_)
> CHPFD_DISTRIBUTE WN      (      _BLK_)
> CHPFD_DISTRIBUTE RHO      (      _BLK_)
>
>
>      real      (kind      =real_kind ),
>      &          dimension  (  2,nconns ),
>      &          intent    (in      )
>      &          :: DELTAUT,DELTAVT,DELTAWT
> CHPFD_DISTRIBUTE DELTAUT      (_STR_,_BLK_)
> CHPFD_DISTRIBUTE DELTAVT      (_STR_,_BLK_)
> CHPFD_DISTRIBUTE DELTAWT      (_STR_,_BLK_)
>
>
>      real      (kind      =real_kind ),
>      &          dimension  (  nconns ),
>      &          intent    (out     )
>      &          :: DELTAUTM,DELTAVTM,DELTAWTM
> CHPFD_DISTRIBUTE DELTAUTM      (      _BLK_)
> CHPFD_DISTRIBUTE DELTAVTM      (      _BLK_)
> CHPFD_DISTRIBUTE DELTAWTM      (      _BLK_)
>
>
>      real      (kind      =real_kind ),
>      &          dimension  (  nconns ),
>      &          intent    (in      )
>      &          :: DDPDXM,DDPDYM,DDPDZM, VOLFLOWM
>
> CHPFD_DISTRIBUTE DDPDXM      (      _BLK_)
> CHPFD_DISTRIBUTE DDPDYM      (      _BLK_)
> CHPFD_DISTRIBUTE DDPDZM      (      _BLK_)
> CHPFD_DISTRIBUTE VOLFLOWM      (      _BLK_)
>
>
>      real      (kind      =real_kind ),      ! CTZ
>      &          dimension  (  nconns )
>      &          :: RHOM
> CHPFD_DISTRIBUTE RHOM      (      _BLK_)      ! CTZ
>
>


---


>
> C      Local variables.
>
>
>      real      (kind      =real_kind )
>      &          :: velmax
>      &          ,rpgs2      !DTR
>

```

C

C

```

>      real      (kind      =real_kind ),
>      &          dimension  (  nnodemax)
>      &          :: SCRATCHN1 ,
>      &          DIVTAUXN    ,DIVTAUYN    ,DIVTAUZN
> CHPFD_DISTRIBUTE SCRATCHN1 (    _BLK_)
> CHPFD_DISTRIBUTE DIVTAUXN  (    _BLK_)
> CHPFD_DISTRIBUTE DIVTAUYN  (    _BLK_)
> CHPFD_DISTRIBUTE DIVTAUZN  (    _BLK_)
> CHPFD_DISTRIBUTE DIVTAUXN  (    _BLK_)
> CHPFD_DISTRIBUTE DIVTAUYN  (    _BLK_)
> CHPFD_DISTRIBUTE DIVTAUZN  (    _BLK_)
>
>
>      real      (kind      =real_kind ),
>      &          dimension  (   nconns )
>      &          :: SCRATCHM1
> CHPFD_DISTRIBUTE SCRATCHM1 (    _BLK_)
>
>      real      (kind      =real_kind ),      ! CTZ
>      &          dimension  (   nconns )
>      &          :: DPDXM    ,   DPDYM    ,   DPDZM
> CHPFD_DISTRIBUTE DPDXM     (    _BLK_)
> CHPFD_DISTRIBUTE DPDYM     (    _BLK_)
> CHPFD_DISTRIBUTE DPDZM     (    _BLK_)
>
>
>      real      (kind      =real_kind ),      ! CTZ
>      &          dimension  (   nnodemax)
>      &          :: CNU      ,   CNV      ,   CNW      ,      ! CTZ
>      &          CNEIU      ,   CNEIV      ,   CNEIW      ! CTZ
> CHPFD_DISTRIBUTE CNU        (    _BLK_)
> CHPFD_DISTRIBUTE CNV        (    _BLK_)
> CHPFD_DISTRIBUTE CNW        (    _BLK_)
> CHPFD_DISTRIBUTE CNEIU      (    _BLK_)
> CHPFD_DISTRIBUTE CNEIV      (    _BLK_)
> CHPFD_DISTRIBUTE CNEIW      (    _BLK_)
>
>      real      (kind      =real_kind ),      ! CTZ
>      &          dimension  (  2,nconns )
>      &          :: CNU      ,   CNVT      ,   CNWT      ,
>      &          CNEIUT      ,   CNEIVT      ,   CNEIWT
> CHPFD_DISTRIBUTE CNU        (_STR_,_BLK_)
> CHPFD_DISTRIBUTE CNVT        (_STR_,_BLK_)
> CHPFD_DISTRIBUTE CNWT        (_STR_,_BLK_)
> CHPFD_DISTRIBUTE CNEIUT      (_STR_,_BLK_)
> CHPFD_DISTRIBUTE CNEIVT      (_STR_,_BLK_)
> CHPFD_DISTRIBUTE CNEIWT      (_STR_,_BLK_)
>
>      real      (kind      =real_kind ),      ! CTZ
>      &          dimension  (   nconns )
>      &          :: CNEIUM    ,   CNEIVM    ,   CNEIWM
> CHPFD_DISTRIBUTE CNEIUM      (    _BLK_)      ! CTZ
> CHPFD_DISTRIBUTE CNEIVM      (    _BLK_)
> CHPFD_DISTRIBUTE CNEIWM      (    _BLK_)
>
>      integer    (kind      =int_kind )      ! CTZ
>      &          :: ii,k1,k2,jj

```

```

>
>
#####
> CTZ *
> CTZ *
>
> C      Compute coefficient CNU (coefficient of node velocity)
>
> C      Note that VOLFLOWM at a connection is outflow for one of the
cells
> C      and inflow for the other(change of sign below).
>
>      CNUT(1,:) = half*(one+ VFXUPW(:))*RHOM(:)*VOLFLOWM(:)
>      CNUT(2,:) =-half*(one- VFXUPW(:))*RHOM(:)*VOLFLOWM(:)
>      CNVT(1,:) = CNUT(1,:)
>      CNVT(2,:) = CNUT(2,:)
>      CNWT(1,:) = CNUT(1,:)
>      CNWT(2,:) = CNUT(2,:)
>
> C      Compute the sum of the products Cneighbor*Uneighbor
>
>      CNEIUT(1,:) = CNUT(2, :)*DELTAUT(2, :)
>      CNEIUT(2,:) = CNUT(1, :)*DELTAUT(1, :)
>      CNEIVT(1,:) = CNVT(2, :)*DELTAVT(2, :)
>      CNEIVT(2,:) = CNVT(1, :)*DELTAVT(1, :)
>      CNEIWT(1,:) = CNWT(2, :)*DELTAWT(2, :)
>      CNEIWT(2,:) = CNWT(1, :)*DELTAWT(1, :)
>
> C      Accumulate from connections to nodes to compute CNU
>
>      call scattert_vn('add', CNU , CNV, CNW,
> &      CNUT, CNVT, CNWT )
>
> C      Compute contribution from outflow boundaries. Note that
> C      current CHAD version uses first order upwinding at
> C      outflow(see solve_cont)
>
> C      Compute volumetric flow and donor density at outflow
> C      boundaries. The volumetric flow, QFLOWOUT, and donor density,
> C      RHOOUTFLOW, are defined where the flow is actually out of the
> C      problem (QFLOWOUT always negative), and are set to zero at all
> C      non-outflow or inflow/outflow boundaries where the flow is
> C      into the problem.
>
>      SCRATCHN1 = zero
>      SCRATCHN1= (UN-UMESH)*AXN_OPEN
> &      + (VN-VMESH)*AYN_OPEN
> &      + (WN-WMESH)*AZN_OPEN
>      WHERE (P_OPEN .GE.hugenum .OR.
> C &      SCRATCHN1.GE.zero .OR. ! CTZ
> C &      NODETYPE .EQ.ntp_free .OR.
> &      NODETYPE .EQ.ntp_free_src.OR.
> &      NODETYPE .EQ.ntp_inflow. OR. ! CTZ
> &      NODETYPE .EQ.ntp_intrfc
> &      )
>      SCRATCHN1 = zero
>      ELSEWHERE

```

```

>          SCRATCHN1 = RHO * SCRATCHN1
>      ENDWHERE
>
> C          SCRATCHN1 is used below with a negative sign because
> C          AXN_OPEN etc are negative
> C           $d(\rho \cdot u)/dt = \text{momentum(in)} - \text{momentum(out)} + \dots$ 
>
>          CNU = VOL*RHO/dt - SCRATCHN1 + CNU
>          CNV = VOL*RHO/dt - SCRATCHN1 + CNV
>          CNW = VOL*RHO/dt - SCRATCHN1 + CNW
>
> C          Accumulate from connections to nodes to compute CNEIU:
> C          (sum of Cneighbor*Uneighbor)
>
>          call scattert_vn('add', CNEIU , CNEIV, CNEIW,
> &          CNEIUT, CNEIVT, CNEIWT )
>
> c          call difft_tau(DELTAUT,DELTAVT,DELTAWT,
> c &          AMUL,AMUT,AMULM,AMUTM,
> c &          TAUDOTAXM=DELTAUTM,TAUDOTAYM=DELTAVTM,TAUDOTAZM=DELTAWTM,
> c &          DIVTAUXN=DIVTAUXN, DIVTAUYN=DIVTAUYN, DIVTAUZN=DIVTAUZN)
>
> c          SRCMOMU does not change for steady state, only transients,
and we exclude for now
>
> c          CNEIU = half*(CNEIU + SRCMOMU)/CNU
> c          CNEIV = half*(CNEIV + SRCMOMV)/CNV
> c          CNEIW = half*(CNEIW + SRCMOMW)/CNW
>
>          CNEIU = half*CNEIU/CNU
>          CNEIV = half*CNEIV/CNV
>          CNEIW = half*CNEIW/CNW
>
>          CNU = half*VOL/CNU
>          CNV = half*VOL/CNV
>          CNW = half*VOL/CNW
>
> C      To Compute averages transfer from nodes to connection terminuses
>
>          call gathert_vn(CNU, CNV, CNW, CNUT, CNVT, CNWT)
>          call gathert_vn(CNEIU, CNEIV, CNEIW, CNEIUT, CNEIVT, CNEIWT)
>
> C      Compute averages at connections
>
>          CNUM = CNUT(1,:) + CNUT(2,:)
>          CNVM = CNVT(1,:) + CNVT(2,:)
>          CNWM = CNWT(1,:) + CNWT(2,:)
>
>          CNEIUM = CNEIUT(1,:) + CNEIUT(2,:)
>          CNEIVM = CNEIVT(1,:) + CNEIVT(2,:)
>          CNEIWM = CNEIWT(1,:) + CNEIWT(2,:)
>
> C      Compute velocities at median mesh points (tilda velocs)
>
>          DELTAUTM = CNEIUM - CNUM * DDPDXM
>          DELTAVTM = CNEIVM - CNVM * DDPDYM

```

```

> DELTAWTM = CNEIWM - CNWM * DDPDZM
>
> if(flagnofluxm) then
>     WHERE (NOFLUXM)
>         DELTAUTM=zero
>         DELTAVTM=zero
>         DELTAWTM=zero
>     ENDWHERE
> endif
>
>
>
> *****
>
>     end subroutine cellface_tilde_change_ctz_cont
>
>     subroutine cellface_coeff()
>
>
> #####
>
> C     Purpose:
>
>     Recomputes the portion of the reformulated tilde velocity
> c     that multiplies pressure.
>
> C     Input Variables:
>
> C         UUT          = Intermediate x-direction velocity gathered
at
> C                     connection terminuses.
> C         VVT          = Intermediate y-direction velocity gathered
at
> C                     connection terminuses.
> C         WWT          = Intermediate z-direction velocity gathered
at
> C                     connection terminuses.
> C         * Intermediate values as updated in the coupling of the
cont.
> C         equation for the inner-inner cont iterations.
>
> C     Output Variables:
>
> C         CNUM      =
> C         CNVM      =
> C         CNWM      =
>
>
>
> #####
>
> C     Module list and other preliminaries
>
>     use          arrays_glo_module_c          , only:
> &               NODETYPE          , X, Y, Z, P,
> &               STRSXX          , STRSXY          , STRSXZ          ,
> &               STRSYY          , STRSYZ          , STRSZZ          ,
> &               DELTAL          , UMESH          , VMESH          , WMESH
.

```

```

>      &          AXM          ,AYM          ,AZM          ,
>      &          VOLSWPTRATEM, UWALL, VWALL, WWALL, VOL,
>      &          RHO , VOL , MASSOLD , UOLD , VOLD , WOLD,
! CTZ
>      &          AXN_OPEN , AYN_OPEN , AZN_OPEN,
>      &          U, UT, V, VT, W, WT, NODEST,RHO          ! CTZ
>      use          hydro_module_c          ,only:
>      &          AMUL , AMUT , AMULM, AMUTM, RHOT, NOFLUXM,
! CTZ
>      &          flagnofluxm,RHOM,CNUM,CNVM,CNWM,          ! DTR
>      &          UTILDEM,VTILDEM,WTILDEM,VOLFLOWM
>      use          nodetypes_module_c          ,only:
>      &          ntp_inflow
>      use          options_module_c          ,only:
>      &          flowtype
>      use          physics_module_c          ,only:
>      &          P_OPEN, SRCMOMU,SRCMOMV,SRCMOMW
>      use          physdata_module_c          ,only:
>      &          ftlag
>      use          physics_module_c          ,only:
>      &          USOURCE, VSOURCE, WSOURCE
>      &          ,SRCMOMU , SRCMOMV , SRCMOMW , P_OPEN          !
CTZ
>      use          timedata_module_c          ,only:
>      &          dt
>      &          , ncyc          !
CTZ
>      use          gather_module_d          ,only:
>      &          gathert_sn ,gathert_vn
>      use          scatter_module_d          ,only:
>      &          scattert_vn          ! CTZ
>      use          grads_module_d          ,only:
>      &          divn_strsn
>      &          ,gradt_sm , diff_t_tau,gradt_sn
>      use          nodetypes_module_c          ! CTZ
>
>      implicit none
>
>
>
##### C
>
>      Local variables.
>
>
>
##### C
>
>      Global variables being passed through the argument list.
>
>
>      real          (kind          =real_kind ),          ! CTZ
>      &          dimension (          nnodemax)
>      &          :: CNU, CNV, CNW,          ! CTZ
>      &          SCRATCHN1,          ! CTZ
>      &          UN,VN,WN
>      CHPDFD_DISTRIBUTE UN          (          _BLK_)
>      CHPDFD_DISTRIBUTE VN          (          _BLK_)
>      CHPDFD_DISTRIBUTE WN          (          _BLK_)

```



```

> CHPFD_DISTRIBUTE CNU      (      _BLK_)
> CHPFD_DISTRIBUTE CNV      (      _BLK_)
> CHPFD_DISTRIBUTE CNW      (      _BLK_)
> CHPFD_DISTRIBUTE SCRATCHN1 (      _BLK_)
>
>      real      (kind      =real_kind  ),      !      CTZ
>      &          dimension  ( 2,nconns  )
>      &          :: CNUT , CNVT , CNWT
> CHPFD_DISTRIBUTE CNUT      (_STR_,_BLK_)
> CHPFD_DISTRIBUTE CNVT      (_STR_,_BLK_)
> CHPFD_DISTRIBUTE CNWT      (_STR_,_BLK_)
>
>
>      integer  (kind      =int_kind  )      ! CTZ
>      &          :: istop      ,ii,k1,k2
>
>      C          Nonintuitive local variables are defined as follows:
>
>
>
>
> #####
>
>
>      CNUT(1,:) = half*(one+ VFXUPW(:))*RHOM(:)*VOLFLOWM(:)
>      CNUT(2,:) =-half*(one- VFXUPW(:))*RHOM(:)*VOLFLOWM(:)
>
>      CNVT(1,:) = CNUT(1,:)
>      CNVT(2,:) = CNUT(2,:)
>      CNWT(1,:) = CNUT(1,:)
>      CNWT(2,:) = CNUT(2,:)
>
>
>      C          Accumulate from connections to nodes to compute CNU
>
>
>      call scattert_vn('add', CNU, CNV, CNW,
>      &                  CNUT, CNVT, CNWT )
>
>
>      C          Compute contribution from outflow boundaries. Note that
>      C          current CHAD version uses first order upwinding at
>      C          outflow(see solve_cont)
>
>
>      C          Compute volumetric flow and donor density at outflow
>      C          boundaries. The volumetric flow, QFLOWOUT, and donor density,
>      C          RHOOUTFLOW, are defined where the flow is actually out of the
>      C          problem (QFLOWOUT always negative), and are set to zero at all
>      C          non-outflow or inflow/outflow boundaries where the flow is
>      C          into the problem.
>
>
>
>      SCRATCHN1 = zero
>      SCRATCHN1= (UN-UMESH)*AXN_OPEN
>      &          +(VN-VMESH)*AYN_OPEN
>      &          +(WN-WMESH)*AZN_OPEN
>
>      WHERE(P_OPEN      .GE.hugenum      .OR.
>      C      &          SCRATCHN1.GE.zero      .OR.      !      CTZ
>      C      &          NODETYPE .EQ.ntp_free      .OR.
>      &          NODETYPE .EQ.ntp_free_src.OR.

```

```

> &      NODETYPE .EQ.ntp_inflow .OR.
> &      NODETYPE .EQ.ntp_intrfc
> &      )
>      SCRATCHN1 = zero
> ELSEWHERE
>      SCRATCHN1 = RHO * SCRATCHN1
> ENDWHERE
>
>      CNU = VOL*RHO/dt - SCRATCHN1 +CNU
>      CNV = VOL*RHO/dt - SCRATCHN1 +CNV
>      CNW = VOL*RHO/dt - SCRATCHN1 +CNW
>
>      CNU = half*VOL/CNU
>      CNV = half*VOL/CNV
>      CNW = half*VOL/CNW
>
> C      To Compute averages transfer from nodes to connection terminuses
>
>      call gathert_vn(CNU, CNV, CNW, CNUT, CNVT, CNWT)
>
> C      Compute averages at connections
>
>      CNUM = CNUT(1,:) + CNUT(2,:)
>      CNVM = CNVT(1,:) + CNVT(2,:)
>      CNWM = CNWT(1,:) + CNWT(2,:)
>
>
> *****
>
>      end subroutine cellface_coeff
>
>
3094d4353
<
3173a4433
> &      ,X,Y,Z,NODEST
3202d4461
<
3265,3266c4524
< &      SCRATCHN1 ,
< &      UTILDEN ,VTILDEN ,WTILDEN
---
> &      SCRATCHN1
3271,3273d4528
< CHPFD_DISTRIBUTE UTILDEN ( _BLK_)
< CHPFD_DISTRIBUTE VTILDEN ( _BLK_)
< CHPFD_DISTRIBUTE WTILDEN ( _BLK_)
3297a4553,4555
>
>      integer (kind =int_kind )
> &      :: k1,k2,ii
3311d4568
<
3363a4621
>
3492a4751

```

```

>
3657d4915
<
4397a5656
>      print *, 'cellface tc3'
4567d5825
<
4949a6208
>      &      VFXUPW,
5038a6298
>      &      ,NODEST ! CDTR
5124a6385,6388
>      real      (kind      =real_kind  ),
>      &      dimension  (   nconns  )
>      &      :: VFXUPW      !   CTZ
>  CHPDFD_DISTRIBUTE VFXUPW      (      _BLK_)      ! CTZ
5154a6419,6420
>      integer      (kind      =int_kind  )
>      &      :: ii,k1,k2
5159a6426,6455
>  C      If first order upwind (iup_uvw =1) then compute UM,VM,WM as
below
>  C
>
>  c      if(ictz.eq.1)then
>  c      open(unit=58,file='courantm-ctz.dat')
>  c      else
>  c      open(unit=58,file='courantm-upw.dat')
>  c      end if
>  c      do ii=1,nconns
>  c      k1=nodeest(1,ii); k2=nodeest(2,ii)
>  c      write(58,*)k1,k2,'courant-m',COURANTM(ii)
>  c      end do
>
>
>      if(iup_uvw.eq.1) then      ! CTZ
>      VFXUPW =one
>      WHERE(COURANTM.LT.zero)
>      VFXUPW = -one
>      ENDWHERE
>
>  c      do ii=1,nconns
>  c      k1=nodeest(1,ii); k2=nodeest(2,ii)
>  c      write(58,*)k1,k2,'VFXUPW',VFXUPW(ii)
>  c      end do
>
>      UM = half*(UT(1,:)*(one+ VFXUPW) + UT(2,:)*(one-VFXUPW))
>      VM = half*(VT(1,:)*(one+ VFXUPW) + VT(2,:)*(one-VFXUPW))
>      WM = half*(WT(1,:)*(one+ VFXUPW) + WT(2,:)*(one-VFXUPW))
>
>      else      ! CTZ
5489c6785
<
---
>      close(58)
5495a6792,6793
>

```

```
>          end if          !ctz  upwind
```

CHAd_MAIN

Contains modifications for the use of PETSc, and monitoring of data.

```
27a28
>
58a60,68
>      use          anl_data_module_c          ,only:
>      &            matrix_solv, ictz, iup_rho, iup_uvw,matrix_osolv,
>      &            plot_res, plot_mon,
>      &            norm_fact_u,
>      &            norm_fact_v,
>      &            norm_fact_w,
>      &            norm_fact_t,
>      &            norm_fact_p
>      use          inout_module_d
231a242,253
>
>
*****
>
> CDTR  Initialize PETSc
>      if((matrix_solv).or.(matrix_osolv)) then
>          print *, 'initializing petsc'
>          call petsc_chad_init
>      end if
>
>
*****
>
>
498a521,574
> C      Set up file for monitoring points
>
>      if(plot_mon)then
>          call  open_file(lun=55,file='monitors.dat',iostat=iostatus,
>      &                  status='unknown', form='formatted')
>          Inout_String=' '
>          write(Inout_String,"('  ncyc          U-VEL          '
>      &                  'V-VEL          '
>      &                  'W-VEL          '
>      &                  ' PR          '
>      &                  ' time'
>      &                  )")
>          call write_string(Inout_String,.false.,55)
>
>          call close_file(lun=55)
>      end if
>
>
*****
>
```

C

C

C

```

> C      Set up file for residual monitoring
>
>      if (plot_res) then
>
>          call open_file(lun=56,file='residuals.dat',iostat=iostat,
>      &                  status='unknown', form='formatted')
>          Inout_String=' '
>          write(Inout_String,"('  ncyc          U-RES          '
>      &                  'V-RES          '
>      &                  'W-RES          '
>      &                  ' P-RES          '
>      &                  ' T-RES          '
>      &                  ' time'
>      &                  )")
>
>          call write_string(Inout_String,.false.,56)
>
>          call close_file(lun=56)
>
>          norm_fact_u=tiny
>          norm_fact_v=tiny
>          norm_fact_w=tiny
>          norm_fact_t=tiny
>          norm_fact_p=tiny
>
>      end if
>
>
>
> *****
>
> C      Setup monitoring locations ! DTR
>
>      call User_monitor()      ! DTR
>
>
> *****
>
523a600,606
>
>
> *****
> CDTR  Close PETSc gracefully
>
>      if((matrix_solve).or.(matrix_solve))then
>          call petsc_finalize
>      end if

```

```

<      &      nparmax
---
>      &      nparmax      ,nnodemaxdtr  ! DTR
56c57,58
<      &      :: nconns      ,nelemax      ,nnodemax      ,nparmax
---
>      &      :: nconns      ,nelemax      ,nnodemax      ,nparmax,
>      &      nnodemaxdtr      ! DTR

```

DRIVER

Contains some experimental work related to the use of alternate boundary conditions..

```

34a35
>
43a45,47
>
>      use      anl_data_module_c      ,only:
>      &      alt_bc
82,84c86
<
<      call read_msh_or_rst_dist(distributed)
<
---
>
112a115,119
>
>      if (alt_bc) then
>          call work_bcs ! DTR
>      end if
>

```

EOS_MODULE_D66a67

Adds an incompressible equation of state

```

87a89,90
>      use      bcdata_module_c      ,only:
>      &      rho_in      ! CDTR needed for incompressible eos to
set to inlet density
92c95
<
---
>      integer      (kind=int_kind) ii
455a459,465
>      CDTR      ! incompressible
>          case('incompress')
>
>
>      CVOLD=MERGE(DHDTP,CVOLD,MATNON.EQ.Eosdata(imat)%matno)
>
>

```



```

> cdtr for incompressible
>         type      (Incompress_Type      )
>         &          :: Incompress
> cdtr -----

```

HYDRO

Primarily contains modifications for the reformulated continuity equation.

```

179c179
< C      MASSFRAC      = Macro species' mass fractions at time-level n.
---
> C      MASSFRAC      = Macro species mass fractions at time-level n.
679a680
>
688a690,691
>         use          anl_data_module_c          ,only:
>         &            itmaxout, ictz, matrix_solv, plot_res
734a738,739
>         use          solve_hydro_module_c        ,only:      ! DTR
>         &            RESUL,RESVL,RESWL,RESTL,RESCL
754c759
<         &            :: baddata
---
>         &            :: baddata,dtrlog
760c765,766
<         &            npmin          ,nrhomin          ,ntmin
---
>         &            npmin          ,nrhomin          ,ntmin,
>         &            k1,k2,ii !dtr
768d773
<
778a784,793
> c      real          (kind          =real_kind  ),      !CDTR
> c      &             dimension      ( nconns)
> c      &             :: DPDXM,DPDYM,DPDZM
>
>
>         real          (kind          =real_kind  ),      !CDTR
>         &             dimension      (2, nconns)
>         &             :: TMPT1,TMPT2,TMPT3
>
>
814a830
>
821a838,840
>         if(ictz.eq.1)then
>             call zero_walls(.true.)
>         end if
906c925
< C      density] because they won't be exactly consistent with
the
---

```



```

> C          density] because they will not be exactly consistent with
the
1230c1249,1253
<
---
> c          open(unit=55,file='thrmal-cond.dat')
> c          do ii=1,nnodemax
> c              write(55,*)ii,THRMCOND(ii)
> c          end do
> c          stop
1256c1279
< C          wall.  But we don't care about this because
---
> C          wall.  But we do not care about this
because
1331a1355,1357
>          if(ictz.eq.1) then
>          P=POLD          !CTZ-DTR
>          end if
1335a1362,1366
>          if(ictz.eq.1)then
>          U=UOLD          !CTZ
>          V=VOLD          !CTZ
>          W=WOLD          !CTZ
>          end if
1343d1373
<
1375,1376c1405
<          call gathert_vn(U,V,W,
<          & UT,VT,WT)
---
>          call gathert_vn(U,V,W,UT,VT,WT)
1404,1405c1433,1434
<          call gradbcm_sn(P,P_OPEN,SCRATCHM1,
<          & DPDX,DPDY,DPDZ)
---
>
>          call gradbcm_sn(P,P_OPEN,SCRATCHM1,DPDX,DPDY,DPDZ)
1437c1466
<          SCRATCHN1=DPDX
---
>          SCRATCHN1=DPDX
1496,1497c1525
<          call gathert_vn(RESU,RESV,RESW,
<          & SCRATCHT1,SCRATCHT2,SCRATCHT3)
---
>          call gathert_vn(RESU,RESV,RESW,SCRATCHT1,SCRATCHT2,SCRATCHT3)
1499,1510c1527,1563
<          if(tilde_scheme.eq.'highmach') then
<              call cellface_tilde(P,PT,
<              & RHOOLD,RHOOLDT,SCRATCHT1,SCRATCHT2,SCRATCHT3,
<              & UTILDEM,VTILDEM,WTILDEM,VOLFLOWM)
<          else
<              call cellface_tilde(P,U,V,W,
<              & PT,UT,VT,WT,
<              & SCRATCHN1,SCRATCHN2,SCRATCHN3,
<              & POLD,RHOOLD,

```

```

<      &      POLDT,RHOOLDT,SOUNDOLDT,SCRATCHT1,SCRATCHT2,SCRATCHT3,
<      &      UTILDEM,VTILDEM,WTILDEM,VOLFLOWM)
<      endif
---
> C      Compute guess VOLFLOWM needed at cycle one for the computation
> C      of "tilde" velocity wiith CTZ approach
> C      Initialize XMASSV to zero
>
> CDTR This is step #1 of the reformulated continuity equation in my
> notes
> CDTR Quick and dirty check to reinit VOLFLOWM on a restart
>      if(ictz.eq.1)then      !cdtr
>      if(
>      &      (ncyc.eq.1)
>      &      .or.
>      &      ((Global_Sum(VOLFLOWM).eq.0).and.restart)
>      &      ) then      ! CTZ
>      VOLFLOWM=half*
>      &      (
>      &      (UT(1,:)+UT(2,:))*AXM+
>      &      (VT(1,:)+VT(2,:))*AYM+
>      &      (WT(1,:)+WT(2,:))*AZM
>      &      )
>      else
>      VOLFLOWM = VOLFLOLDM
>      endif
>      end if ! cdtr
>
>      if(ictz.ne.1) then      ! ctz actually commented the below out
>      if(tilde_scheme.eq.'highmach') then
>      call cellface_tilde(P,PT,
>      &      RHOOLD,RHOOLDT,SCRATCHT1,SCRATCHT2,SCRATCHT3,
>      &      UTILDEM,VTILDEM,WTILDEM,VOLFLOWM)
>      else
>      call cellface_tilde(P,U,V,W,
>      &      PT,UT,VT,WT,
>      &      SCRATCHN1,SCRATCHN2,SCRATCHN3,
>      &      POLD,RHOOLD,
>      &      POLDT,RHOOLDT,SOUNDOLDT,SCRATCHT1,SCRATCHT2,SCRATCHT3,
>      &      UTILDEM,VTILDEM,WTILDEM,VOLFLOWM)
>      endif
>      end if !ictz=0
1624d1676
< C      Start the outer-iteration loop.
1625a1678,1682
> C      Start the outer-iteration loop.
> c      if(matrix_solv) then
> c      iterout_use=7
> c      limit_semiimplicit=14
> c      else
1631a1689,1690
> cdtr not commenting this out because not sure why CTZ put this in
> cdtr      iterout_use=itmaxout      !      CTZ
1632a1692
> c      end if ! matrix_solv
1641a1702,1717
> cdtr      this is step two corresponding to ctz approach

```



```

366a367
>
422c423,426
<      &          WM          ,WTILDEM
---
>      &          WM          ,WTILDEM      ,
>      &          CNUM , CNVM , CNWM          ,          ! CTZ-DTR
>      &          CNEIUM , CNEIVM , CNEIWM      ,! CTZ-DTR
>      &          VFXUPW          ! CTZ-DTR
616a621,623
>      &          , VFXUPW          ! CTZ-DTR
>      &          , CNUM , CNVM , CNWM          ! CTZ-DTR
>      &          , CNEIUM , CNEIVM , CNEIWM      ! CTZ-DTR
630a638,644
> CHPFD_DISTRIBUTE VFXUPW      (      _BLK_)      ! CTZ-DTR
> CHPFD_DISTRIBUTE CNUM      (      _BLK_)      ! CTZ-DTR
> CHPFD_DISTRIBUTE CNVM      (      _BLK_)      ! CTZ-DTR
> CHPFD_DISTRIBUTE CNWM      (      _BLK_)      ! CTZ-DTR
> CHPFD_DISTRIBUTE CNEIUM      (      _BLK_)      ! CTZ-DTR
> CHPFD_DISTRIBUTE CNEIVM      (      _BLK_)      ! CTZ-DTR
> CHPFD_DISTRIBUTE CNEIWM      (      _BLK_)      ! CTZ-DTR
912a927,933
>      &          VFXUPW      (      nconns      ),          ! CTZ-DTR
>      &          CNUM      (      nconns      ),          ! CTZ-DTR
>      &          CNVM      (      nconns      ),          ! CTZ-DTR
>      &          CNWM      (      nconns      ),          ! CTZ-DTR
>      &          CNEIUM      (      nconns      ),          ! CTZ-DTR
>      &          CNEIVM      (      nconns      ),          ! CTZ-DTR
>      &          CNEIWM      (      nconns      ),          ! CTZ-DTR
1104a1126,1132
>      &          CNUM          , !CTZ-DTR
>      &          CNVM          , !CTZ-DTR
>      &          CNWM          , !CTZ-DTR
>      &          CNEIUM        , !CTZ-DTR
>      &          CNEIVM        , !CTZ-DTR
>      &          CNEIWM        , !CTZ-DTR
>      &          VFXUPW        , !CTZ-DTR

```

MATRIX_COEFFS

This routine modularizes some changes for calculating matrix coefficients. It was not used in the final edition of the code, but is included here for reference since it showed up in the source differencing. It is not guaranteed that this routine works.

```

*DK matrix_coeffs
      subroutine matrix_coeffs

C#####
#

C      Purpose:

```

```

C          For SIMPLE solver

C          Constructs coeffs for a matrix system of Ax=b form.

C          For Newton-Krylov solver:

C          Not used

C      Input Variables:

C          CNUM          =
C          CNVM          =
C          CNWM          =

C          RHO           = Current density.
C          RHOM          = Density at median-mesh boundary.

C      Output Variables:

C          For SIMPLE solver:

C          A0T           = Off diagonal entries of the matrix A
C          A0            = Main diagonal entries of the matrix A
C          AT            = Filled off diagonal entries

C#####
C#

C      Module list and other preliminaries.

#include "macros.HH"

        use          Kinds_module_c
        use          arrays_glo_module_c
        use          constants_module_c
        use          dimensions_module_c
        use          hydro_module_c
        use          physdata_module_c
        use          physics_module_c
#if SIMPLE
        use          solve_cont_module_c
#else /* SIMPLE */
        use          solve_hydro_module_c
#endif /* SIMPLE */
        use          gather_module_d
        use          scatter_module_d
        use          timedata_module_c
        use          anl_data_module_c
        use          symmetrize_module_d      ,only:
&          symmetrize_sl
        use          bcs_dep_module_d        ,only:
&          bc_s
        implicit     none

C#####
C#

```

```

C      Other global and local variables.

C_____
-

C      Global variables being passed through the argument list.

      integer      (kind      =int_kind  )
      &            :: ptc_its                ! # iterations ptc
requires
      &            ,ii, k1, k2                ! upwinding flag, loop
counter,indices

      real          (kind      =real_kind ),
      &            dimension   (2, nconns)
      &            :: DRESUT, DRESVT, DRESWT

      real          (kind      =real_kind ),
      &            dimension   (nnodemax)
      &            :: DRESU, DRESV, DRESW

CHPFD_DISTRIBUTE DRESUT      (      _BLK_ )
CHPFD_DISTRIBUTE DRESVT      (      _BLK_ )
CHPFD_DISTRIBUTE DRESWT      (      _BLK_ )
CHPFD_DISTRIBUTE DRESU       (      _BLK_ )
CHPFD_DISTRIBUTE DRESV       (      _BLK_ )
CHPFD_DISTRIBUTE DRESW       (      _BLK_ )

C_____
-

C#####
#

C
*****

CDTR  Compute the portion of the derivative of the residual with
CDTR  respect to pressure that arises from the dependence of
CDTR  the median mesh velocity on pressure

      SCRATCHM1 = zero

      SCRATCHM1 = one /
&          ( (XT(2,:)-XT(1,:))**2
&          + (YT(2,:)-YT(1,:))**2
&          + (ZT(2,:)-ZT(1,:))**2
&          )

      if ( flagnofluxm ) then
          WHERE(nofluxm)
              CNUM = zero
              CNVM = zero
              CNWM = zero
          ENDWHERE

```

```

        end if

DRESUT(1,:)=(XT(2,:)-XT(1,:))*AXM(:)*RHOM(:)*CNUM(:)*SCRATCHM1
DRESVT(1,:)=(YT(2,:)-YT(1,:))*AYM(:)*RHOM(:)*CNVM(:)*SCRATCHM1
DRESWT(1,:)=(ZT(2,:)-ZT(1,:))*AZM(:)*RHOM(:)*CNWM(:)*SCRATCHM1

DRESUT(2,:)=DRESUT(1,:)
DRESVT(2,:)=DRESVT(1,:)
DRESWT(2,:)=DRESWT(1,:)

A0T(1,:)=DRESUT(1,:)+DRESVT(1,:)+DRESWT(1,:)
A0T(2,:)=A0T(1,:)

CDTR  Gather volume from nodes to connections and store it in the
SCRATCHT1

        call gathert_sn(VOL, SCRATCHT1)
A0T(1,:)=(dt/SCRATCHT1(1,:))*A0T(1,:)
A0T(2,:)=(dt/SCRATCHT1(2,:))*A0T(2,:)

CDTR  Accumulate from connections to nodes

        call scattert_vn('add',DRESU,DRESV,DRESW,DRESUT,DRESVT,DRESWT)

A0=(dt/VOL)*(DRESU+DRESV+DRESW)

c      A0 = A0 + DRHODPT+DRHODTP*CBIG

A0T(1,:)=A0T(1,:)+A0
A0T(2,:)=A0T(2,:)+A0

C      The following may eventually require modifications to A0 instead
of RDRESCDP

c#if PRESSURE_BASED
c      if(index(intrfc_opt,'pequil').gt.0) then
c          call intrfc_s('volume',QN=RDRESCDP)
c      endif
c#endif /* PRESSURE_BASED */

C      Compute the reciprocal of the derivative of residual

#if PRESSURE_BASED
c      RDRESCDP =one/SIGN(MAX(ABS(RDRESCDP),tiny),RDRESCDP)
#endif /* PRESSURE_BASED */

#if PRESSURE_BASED
c      if(flagpsource) then
c          call bc_s('change',PSOURCE,RDRESCDP)
c      endif
#endif /* PRESSURE_BASED */

#if PRESSURE_BASED

c      RDRESCDP=SIGN(MIN(ABS(RDRESCDP),
c      & 10.0_real_kind*DELTAPMAX),

```

```

c      &                      /MAX (ABS (RESCL), tinyum)
c      &                      ),
c      &                      RDRESCDP
c      &                      )

c      RDRESCDP=MAX (RDRESCDP, zero)

c      WHERE (NODETYPE.LT.0)
c      RESCL =zero
c      RDRESCDP=zero
c      ENDWHERE

#endif /* PRESSURE_BASED */

c      if(geometry.ne.'3-D') then
#endif PRESSURE_BASED
c      call symmetrize_s1(RDRESCDP)
#endif /* PRESSURE_BASED */
c      end if

c      end if
C
*****

      end

```

PETSC_CHAD_FINALIZE

Closes PETSc nicely.

```

      subroutine petsc_chad_finalize

#include "/home/cluster3/drockken/petsc-2.1.6/include/finclude/petsc.h"

      integer IERR

      call petscfinalize(PETSC_NULL_CHARACTER, IERR)

      end

```

PETSC_CHAD_INIT

Initializes PETSc.

```

      subroutine petsc_chad_init

#include "/home/cluster3/drockken/petsc-2.1.6/include/finclude/petsc.h"

      integer IERR

```



```

call petscinitialize(PETSC_NULL_CHARACTER,IERR)

end

```

PHYSICS_MODULE_C

```

143a144
>
168a170,171
>      public      SRCMOMU      , SRCMOMV      , SRCMOMW
>
205c208,209
<      &          USOURCE      , VSOURCE      , WSOURCE
---
>      &          USOURCE      , VSOURCE      , WSOURCE      ,
>      &          SRCMOMU      , SRCMOMV      , SRCMOMW      ! CTZ
222a227,229
> CHPFD_DISTRIBUTE SRCMOMU      (      _BLK_)      ! CTZ
> CHPFD_DISTRIBUTE SRCMOMV      (      _BLK_)      ! CTZ
> CHPFD_DISTRIBUTE SRCMOMW      (      _BLK_)      ! CTZ
478a486,488
>      &          SRCMOMU      (      nnodemax),      ! CTZ
>      &          SRCMOMV      (      nnodemax),      ! CTZ
>      &          SRCMOMW      (      nnodemax),      ! CTZ
579a590,592
>      &          SRCMOMU      ,      ! CTZ
>      &          SRCMOMV      ,      ! CTZ
>      &          SRCMOMW      ,      ! CTZ

```

READ_INPUT

Reads all the new variables from the input file; sets their default values. Some of this stuff gets brought in from a different ANL routine and is included here as a result of the migration between CHAD 3.0 and CHAD_UIUC.

```

646a647
>
651a653
>      use          anl_data_module_c
716a719,810
>      logical      (kind          =log_kind      )
>      &      :: adb_wall_default      = .false.
>      &      ,matrix_solv_default      = .false.      ! CTZ
>      &      ,matrix_osolv_default= .false.      ! DTR
>      &      ,plot_res_default      = .false.      ! DTR
>      &      ,plot_mon_default      = .false.      ! DTR
>      &      ,float_rtol_default      = .false.      ! DTR
>      &      ,porous_media_default= .false.      ! DTR
>      &      ,use_urfp_default      = .false.      ! DTR
>      &      ,use_urfm_default      = .false.      ! DTR

```

```

>      &      ,use_urftke_default  = .false.      !      DTR
>      &      ,use_urfeps_default  = .false.      !      DTR
>      &      ,use_urft_default    = .false.      !      DTR
>      &      ,alt_bc_default      = .false.      !      DTR
>      &      ,debug_default        = .false.
>      &      ,ipnext_default       = .true.
>      &      ,res_fix_default      = .true.
>      &      ,th_in_default        = .false.
>      &      ,th_wall_default      = .false.
>
>      integer      (kind      =int_kind      )
>      &      ::
>      &      ,ibcs_eps_default      = izero
>      &      ,ibcs_p_default        = izero
>      &      ,ibcs_t_default        = izero
>      &      ,ibcs_tke_default      = izero
>      &      ,ibcs_uvw_default      = izero
>      &      ,idifft_s_default      = izero
>      &      ,idifft_tau_default    = izero
>      &      ,idtfluid_default      = izero
>      &      ,igradltdt_st_default  = izero
>      &      ,ihydro_seq_default    = izero
>      &      ,ikepsilon_default     = izero
>      &      ,imass_tmp_default     = izero
>      &      ,imass_uvw_default     = izero
>      &      ,irhom_default         = izero
>      &      ,isolve_prs_default    = izero
>      &      ,isolve_tmp_default    = izero
>      &      ,isolve_uvw_default    = izero
>      &      ,itilde_cons_default   = izero
>      &      ,itilde_uvw_default    = izero
>      &      ,itmaxout_default      = izero
>      &      ,itmaxoutt_default     = izero
>      &      ,itmaxprs_default      = izero
>      &      ,itmaxtmp_default      = izero
>      &      ,itmaxuvw_default      = izero
>      &      ,iup_rho_default       = izero
>      &      ,iup_tmp_default       = izero
>      &      ,iup_uvw_default       = izero
>      &      ,iup_ph_default        = izero
>      &      ,iutildem_default      = izero
>      &      ,iuvwm_default         = izero
>      &      ,kutildem_default      = izero
>      &      ,kvtildem_default      = izero
>      &      ,kwtildem_default      = izero
>      &      ,ibcsbug_default       = izero
>      &      ,idifft_sbug_default    = izero
>      &      ,idifftaubug_default   = izero
>      &      ,idragbug_default      = izero
>      &      ,ihembug_default       = izero
>      &      ,iysmpbug_default      = izero
>      &      ,ihydrobug_default     = izero
>      &      ,ikebug_default        = izero
>      &      ,imassbug_default      = izero
>      &      ,imeshbug_default      = izero
>      &      ,inode_begin_default   = izero
>      &      ,inode_end_default     = izero

```

```

>      &      ,inode_step_default   = ione
>      &      ,iprsbug_default      = izero
>      &      ,irhombug_default      = izero
>      &      ,itildebug_default     = izero
>      &      ,imatrixbug_default    = izero
>      &      ,i_no_e_default         = izero
>      &      ,iubug_default         = izero
>      &      ,iumbug_default        = izero
>      &      ,iusourcebug_default   = izero
>      &      ,iutildembug_default   = izero
>      &      ,iuvwbug_default       = izero
>      &      ,ivbug_default         = izero
>      &      ,ivmbug_default        = izero
>      &      ,iwbug_default         = izero
>      &      ,iwmbug_default        = izero
>      &      ,ictz_default          = izero
>      &      ,imon_node_u_default   = izero
>      &      ,imon_node_v_default   = izero
>      &      ,imon_node_w_default   = izero
>      &      ,imon_node_t_default   = izero
>      &      ,imon_node_p_default   = izero
>      &      ,imon_node_k_default   = izero
>      &      ,imon_node_e_default   = izero
>      &      ,imodel_id_default     = izero
>
724a819,868
>      real      (kind      =real_kind  )
>      &      ::
>      &      ,gravx_default         = zero
>      &      ,gravy_default         = zero
>      &      ,ravz_default          = zero
>      &      ,rhozero_default       = zero
>      &      ,vexpansion_default    = zero
>      &      ,rhol1_default         = one
>      &      ,rhol2_default         = one
>      &      ,mul1_default          = 1.0e-03
>      &      ,mul2_default          = 1.0e-03
>      &      ,etalset_default       = one
>      &      ,sigma_default         = 1.0e-01
>      &      ,pzero_default         = 1.e5
>      &      ,cpu_endtime_default   = 60.0
>      &      ,cpu_maxtime_default   = bignum
>      &      ,dpcoef_default        = one
>      &      ,epsseps_default       = one
>      &      ,epsflow_default       = smallnum
>      &      ,epsprs_default        = one
>      &      ,epstke_default        = one
>      &      ,epstmp_default        = one
>      &      ,epsuvw_default        = one
>      &      ,fdrag_a_default       = zero
>      &      ,fdrag_b_default       = zero
>      &      ,htc_wall_default      = zero
>      &      ,omegae_default        = one
>      &      ,omegav_default        = one
>      &      ,t_mom_off_default     = bignum
>      &      ,dt_mom_off_default    = bignum
>      &      ,x_exit_default        = zero

```

```

>      &      ,y_exit_default      = zero
>      &      ,z_exit_default      = zero
>      &      ,epsdebug_default     = smallnum
>      &      ,mon_x_default        = zero ! DTR
>      &      ,mon_y_default        = zero ! DTR
>      &      ,mon_z_default        = zero ! DTR
>      &      ,conv_res_u_default   = tinynum ! DTR
>      &      ,conv_res_v_default   = tinynum ! DTR
>      &      ,conv_res_w_default   = tinynum ! DTR
>      &      ,conv_res_t_default   = tinynum ! DTR
>      &      ,conv_res_p_default   = tinynum ! DTR
>      &      ,slv_tol_s_default    = one      ! DTR
>      &      ,slv_tol_v_default    = one ! DTR
>      &      ,urf_t_default        = 0.7
>      &      ,urf_p_default        = 0.3
>      &      ,urf_m_default        = 0.5
>      &      ,urf_tke_default      = 0.85
>      &      ,urf_eps_default      = 0.85
>
1008c1152,1153
<      gmvform_default      ='(1p,10e13.5)'
---
>      gmvform_default      ='(1p,10e17.9)'
> c      gmvform_default      ='(1p,10e13.5)' ! CDTR
1307a1453,1662
> CDTR
> 100 format(
> 1' ...read_anl_input... ',6(a,i5))
> 120 format(a)
> 121 format(/,a)
>
> C      Buoyancy input      ! CTZ
> C      *****
>      call read_var(gravx      , 'gravx'      , gravx_default
,InBuffer)
>      call read_var(gravy      , 'gravy'      , gravx_default
,InBuffer)
>      call read_var(gravz      , 'gravz'      , gravx_default
,InBuffer)
>      call read_var(rhozero     , 'rhozero'     , rhozero_default
,InBuffer)
>
call
read_var(vexpansion, 'vexpansion', vexpansion_default, InBuffer)
>
> C      level set input      ! CTZ
> C      *****
>      call read_var(rhol1      , 'rhol1'      , rhol1_default
,InBuffer)
>      call read_var(rhol2      , 'rhol2'      , rhol2_default
,InBuffer)
>      call read_var(mul1       , 'mul1'       , mul1_default
,InBuffer)
>      call read_var(mul2       , 'mul2'       , mul1_default
,InBuffer)
>      call read_var(etalset     , 'etalset'     , etalset_default
,InBuffer)

```

```

>          call read_var(sigma          , 'sigma'          , sigma_default
, InBuffer)
>
>          write(InOut_String,120) ' CONTROL VARIABLES:'
>          call write_string(InOut_String,tty=.true.)
>          write(InOut_String,121) ' '
>          call write_string(InOut_String,tty=.true.)
>
>
> C      first order upwind input
> C      *****
>
>          call read_var(iup_rho        , 'iup_rho'        , iup_rho_default
, InBuffer)
>          call read_var(iup_tmp        , 'iup_tmp'        , iup_tmp_default
, InBuffer)
>          call read_var(iup_uvw        , 'iup_uvw'        , iup_uvw_default
, InBuffer)
>          call read_var(iup_ph         , 'iup_ph'         , iup_ph_default
, InBuffer)
> C
> C      control variables
> C      *****
>          call read_var(pzero          , 'pzero'          , pzero_default
, InBuffer)
>          call read_var(matrix_solv, 'matrix_solv', matrix_solv_default,
>          & InBuffer)
>          call read_var(matrix_osolv, 'matrix_osolv', matrix_osolv_default,
>          & InBuffer)
>          call read_var(plot_res, 'plot_res', plot_res_default, InBuffer)
>          call read_var(plot_mon, 'plot_mon', plot_mon_default, InBuffer)
>          call read_var(alt_bc, 'alt_bc', alt_bc_default, InBuffer)
>          call read_var(cpu_endtime, 'cpu_endtime', cpu_endtime_default,
>          & InBuffer)
>          call read_var(cpu_maxtime, 'cpu_maxtime', cpu_maxtime_default,
>          & InBuffer)
>          call read_var(dpcoef        , 'dpcoef'        , dpcoef_default
, InBuffer)
>          call read_var(epseps        , 'epseps'        , epseps_default
, InBuffer)
>          call read_var(epsflow       , 'epsflow'       , epsflow_default
, InBuffer)
>          call read_var(epsprs        , 'epsprs'        , epsprs_default
, InBuffer)
>          call read_var(epstke        , 'epstke'        , epstke_default
, InBuffer)
>          call read_var(epstmp        , 'epstmp'        , epstmp_default
, InBuffer)
>          call read_var(epsuvw        , 'epsuvw'        , epsuvw_default
, InBuffer)
>          call read_var(fdrag_a       , 'fdrag_a'       , fdrag_a_default
, InBuffer)
>          call read_var(fdrag_b       , 'fdrag_b'       , fdrag_b_default
, InBuffer)
>          call read_var(htc_wall      , 'htc_wall'      , htc_wall_default
, InBuffer)

```

```

>      call read_var(ibcs_eps      , 'ibcs_eps'      , ibcs_eps_default
,InBuffer)
>      call read_var(ibcs_p       , 'ibcs_p'       , ibcs_p_default
,InBuffer)
>      call read_var(ibcs_t       , 'ibcs_t'       , ibcs_t_default
,InBuffer)
>      call read_var(ibcs_tke     , 'ibcs_tke'     , ibcs_tke_default
,InBuffer)
>      call read_var(ibcs_uvw     , 'ibcs_uvw'     , ibcs_uvw_default
,InBuffer)
>      call read_var(idifft_s     , 'idifft_s'     , idifft_s_default
,InBuffer)
>      call read_var(idifft_tau , 'idifft_tau' , idifft_tau_default ,
>      &
>      InBuffer)
>      call read_var(idtfluid     , 'idtfluid'     , idtfluid_default
,InBuffer)
>      call read_var(igradltdt_st, 'igradltdt_st',igradltdt_st_default
>      &
>      InBuffer)
>      call read_var(ihydro_seq , 'ihydro_seq' , ihydro_seq_default ,
>      &
>      InBuffer)
>      call read_var(ikepsilon   , 'ikepsilon'   , ikepsilon_default
,InBuffer)
>      call read_var(imass_tmp    , 'imass_tmp'    , imass_tmp_default
,InBuffer)
>      call read_var(imass_uvw    , 'imass_uvw'    , imass_uvw_default
,InBuffer)
>      call read_var(ipnext       , 'ipnext'       , ipnext_default
,InBuffer)
>      call read_var(irhom        , 'irhom'        , irhom_default
,InBuffer)
>      call read_var(isolve_prs , 'isolve_prs' , isolve_prs_default ,
>      &
>      InBuffer)
>      call read_var(isolve_tmp , 'isolve_tmp' , isolve_tmp_default ,
>      &
>      InBuffer)
>      call read_var(isolve_uvw , 'isolve_uvw' , isolve_uvw_default ,
>      &
>      InBuffer)
>      call read_var(itolde_cons, 'itolde_cons',itolde_cons_default,
>      &
>      InBuffer)
>      call read_var(itolde_uvw , 'itolde_uvw' , itilde_uvw_default ,
>      &
>      InBuffer)
>      call read_var(itmaxout     , 'itmaxout'     , itmaxout_default
,InBuffer)
>      call read_var(itmaxoutt    , 'itmaxoutt'    , itmaxoutt_default
,InBuffer)
>      call read_var(itmaxprs     , 'itmaxprs'     , itmaxprs_default
,InBuffer)
>      call read_var(itmaxtmp     , 'itmaxtmp'     , itmaxtmp_default
,InBuffer)
>      call read_var(itmaxuvw     , 'itmaxuvw'     , itmaxuvw_default
,InBuffer)
>      call read_var(iutildem     , 'iutildem'     , iutildem_default
,InBuffer)
>      call read_var(iuvwm        , 'iuvwm'        , iuvwm_default
,InBuffer)
>      call read_var(kutildem     , 'kutildem'     , kutildem_default
,InBuffer)

```

```

>      call read_var(kvtilde      , 'kvtilde'      , kvtilde_default
,InBuffer)
>      call read_var(kwtildem     , 'kwtildem'    , kwtildem_default
,InBuffer)
>      call read_var(omegae       , 'omegae'     , omegae_default
,InBuffer)
>      call read_var(omegav       , 'omegav'     , omegav_default
,InBuffer)
>      call read_var(outavs       , 'outavs'     , outavs_default
,InBuffer)
>      call read_var(outfv        , 'outfv'      , outfv_default
,InBuffer)
>      call read_var(outgm        , 'outgm'      , outgm_default
,InBuffer)
>      call read_var(res_fix      , 'res_fix'    , res_fix_default
,InBuffer)
>      call read_var(t_mom_off    , 't_mom_off'  , t_mom_off_default
,InBuffer)
>      call read_var(dt_mom_off   , 'dt_mom_off' , dt_mom_off_default ,
>      &      InBuffer)
>      call read_var(x_exit       , 'x_exit'     , x_exit_default
,InBuffer)
>      call read_var(y_exit       , 'y_exit'     , y_exit_default
,InBuffer)
>      call read_var(z_exit       , 'z_exit'     , z_exit_default
,InBuffer)
>      call read_var(adb_wall     , 'adb_wall'   , adb_wall_default
,InBuffer)
>      call read_var(th_wall      , 'th_wall'    , th_wall_default
,InBuffer)
>      call read_var(th_in        , 'th_in'      , th_in_default
,InBuffer)
>
>      write(InOut_String,121) ' '
>      call write_string(InOut_String, tty=.true.)
>      write(InOut_String,120) ' DEBUG VARIABLES:'
>      call write_string(InOut_String, tty=.true.)
>      write(InOut_String,121) ' '
>      call write_string(InOut_String, tty=.true.)
>
>      call read_var(debug        , 'debug'      , debug_default
,InBuffer)
>      call read_var(epsdebug     , 'epsdebug'   , epsdebug_default
,InBuffer)
>      call read_var(ibcsbug      , 'ibcsbug'    , ibcsbug_default
,InBuffer)
>      call read_var(idifft_sbug  , 'idifft_sbug', idifft_sbug_default,
>      &      InBuffer)
>      call read_var(idifftaubug  , 'idifftaubug', idifftaubug_default,
>      &      InBuffer)
>      call read_var(idragbug     , 'idragbug'   , idragbug_default
,InBuffer)
>      call read_var(ihembug      , 'ihembug'    , ihembug_default
,InBuffer)
>      call read_var(iysmpbug     , 'iysmpbug'   , iysmpbug_default
,InBuffer)

```

```

>      call read_var(ihydrobug      , 'ihydrobug'      , ihydrobug_default
, InBuffer)
>      call read_var(ikebug         , 'ikebug'         , ikebug_default
, InBuffer)
>      call read_var(imassbug       , 'imassbug'       , imassbug_default
, InBuffer)
>      call read_var(imeshbug       , 'imeshbug'       , imeshbug_default
, InBuffer)
>      call read_var(inode_begin, 'inode_begin', inode_begin_default,
>      & InBuffer)
>      call read_var(inode_end   , 'inode_end'   , inode_end_default
, InBuffer)
>      call read_var(inode_step , 'inode_step' , inode_step_default ,
>      & InBuffer)
>      call read_var(iprsbug     , 'iprsbug'     , iprsbug_default
, InBuffer)
>      call read_var(irhombug    , 'irhombug'    , irhombug_default
, InBuffer)
>      call read_var(itildebug   , 'itildebug'   , itildebug_default
, InBuffer)
>      call read_var(imatrixbug , 'imatrixbug' , imatrixbug_default ,
>      & InBuffer)
>      call read_var(i_no_e     , 'i_no_e'     , i_no_e_default
, InBuffer)
>      call read_var(iubug      , 'iubug'      , iubug_default
, InBuffer)
>      call read_var(iumbug     , 'iumbug'     , iumbug_default
, InBuffer)
>      call read_var(iusourcebug, 'iusourcebug', iusourcebug_default,
>      & InBuffer)
>      call read_var(iutildembug, 'iutildembug', iutildembug_default,
>      & InBuffer)
>      call read_var(iuvwbug    , 'iuvwbug'    , iuvwbug_default
, InBuffer)
>      call read_var(ivbug      , 'ivbug'      , ivbug_default
, InBuffer)
>      call read_var(ivmbug     , 'ivmbug'     , ivmbug_default
, InBuffer)
>      call read_var(iwbug      , 'iwbug'      , iwbug_default
, InBuffer)
>      call read_var(iwmbug     , 'iwmbug'     , iwmbug_default
, InBuffer)
>      call read_var(ictz       , 'ictz'       , ictz_default
, InBuffer)
>
> CDTT
>      call read_var(imon_node_u, 'imon_node_u', imon_node_u_default,
>      & InBuffer)
>      call read_var(imon_node_v, 'imon_node_v', imon_node_v_default,
>      & InBuffer)
>      call read_var(imon_node_w, 'imon_node_w', imon_node_w_default,
>      & InBuffer)
>      call read_var(imon_node_t, 'imon_node_t', imon_node_t_default,
>      & InBuffer)
>      call read_var(imon_node_p, 'imon_node_p', imon_node_p_default,
>      & InBuffer)
>      call read_var(imon_node_k, 'imon_node_k', imon_node_k_default,

```



```

>
> read(line,*) Eosdata(imat)%Incompress%cp ,
> & Eosdata(imat)%Incompress%rho0
> Inout_String=' '
>
> write(Inout_String,"(1p,' cp      =',e13.6,';',
> & ' rho0      =',e13.6
> & )"
> & ) Eosdata(imat)%Incompress%cp ,
> & Eosdata(imat)%Incompress%rho0
>
> call write_string(Inout_String,tty=.true.)
>
> C      .      .      .      .      .      .      .
.
>
> C      Read basic species data.
>
> call read_nxtline(line,errorl,Inbuffer)
> read(line,fmt=*,iostat=iostat)
> & Eosdata(imat)%Incompress%nsp ,
> & Eosdata(imat)%Incompress%alpha,
> & Eosdata(imat)%Incompress%beta ,
> & Eosdata(imat)%Incompress%es ,
> & Eosdata(imat)%Incompress%rsc
>
> if(iostat.ne.0 .or.
> & Eosdata(imat)%Incompress%nsp.le.0 .or.
> & Eosdata(imat)%Incompress%nsp.ge.9999.or.
> & Eosdata(imat)%Incompress%rsc.lt.zero
> & ) then
> done_nxtline =.true.
> Eosdata(imat)%Incompress%nsp =1
> Eosdata(imat)%Incompress%alpha=zero
> Eosdata(imat)%Incompress%beta =zero
> Eosdata(imat)%Incompress%es =zero
> Eosdata(imat)%Incompress%rsc =zero
> endif
> Inout_String=' '
> write(Inout_String,"(1p,' nsp      =',i13 ,';',
> & ' alpha    =',e13.6,';',
> & ' beta      =',e13.6,
> & /,' es      =',e13.6,';',
> & ' rsc       =',e13.6
> & )"
> & ) Eosdata(imat)%Incompress%nsp ,
> & Eosdata(imat)%Incompress%alpha,
> & Eosdata(imat)%Incompress%beta ,
> & Eosdata(imat)%Incompress%es ,
> & Eosdata(imat)%Incompress%rsc
> call write_string(Inout_String,tty=.true.)
>
> C      .      .      .      .      .      .      .
.
>
> C      Define the starting and ending location of current
> C      species relative to the global species numbers and

```

```

> C      increment the global number-of-species count.
>
>      Eosdata(imat)%Incompress%ispstrt=nsp+1
>      Eosdata(imat)%Incompress%ispend
>      &      = Eosdata(imat)%Incompress%ispstrt
>      &      +Eosdata(imat)%Incompress%nsp
>      &      -1
>      nsp=nsp+Eosdata(imat)%Incompress%nsp
> C      .      .      .      .      .      .      .      .
>      .
>
> C      Copy the basic scalar data directly under the EOSDATA
> C      structure for easy access.
>
>      Eosdata(imat)%ispstrt=Eosdata(imat)%Incompress%ispstrt
>      Eosdata(imat)%ispend =Eosdata(imat)%Incompress%ispend
>      Eosdata(imat)%nsp    =Eosdata(imat)%Incompress%nsp
>      Eosdata(imat)%alpha  =Eosdata(imat)%Incompress%alpha
>      Eosdata(imat)%beta   =Eosdata(imat)%Incompress%beta
>      Eosdata(imat)%es     =Eosdata(imat)%Incompress%es
>      Eosdata(imat)%rsc    =Eosdata(imat)%Incompress%rsc
>
> C      .      .      .      .      .      .      .      .
>      .
>
> C      Now that we know the number of macro species, allocate
> C      memory to the SPCDATA type.
>
>      allocate(Eosdata(imat)%Incompress
>      &          %Spcdata(Eosdata(imat)%Incompress%nsp)
>      &          )
>
> C      .      .      .      .      .      .      .      .
>      .
>
> C      Read each macro species data at a time and then read
the associated micro species data.
> C
>
>      do isp=1,Eosdata(imat)%Incompress%nsp
>
>      Read macro species ID and the number of associated
> C      micro species.
>
>      if(done_nxtlne) then
>      Eosdata(imat)%Incompress%Spcdata(isp)%id
>      &      =Eosdata(imat)%matname
>      Eosdata(imat)%Incompress%Spcdata(isp)%nsp
>      &      =1
>      else
>      call read_nxtlne(line,errorl,Inbuffer)
>      if(errorl) error=.true.
>      read(line,*) Eosdata(imat)%Incompress
>      &          %Spcdata(isp)%id ,
>      &          Eosdata(imat)%Incompress
>      &          %Spcdata(isp)%nsp
>      endif

```

```

> Inout_String=' '
> write(Inout_String,"(/,' Spc_Id  =',a13  ',';',
> & ' nsp_sub =',i13
> & )"
> & ) Eosdata(imat)%Incompress%Spcdata(isp)%id ,
> & Eosdata(imat)%Incompress%Spcdata(isp)%nsp
> call write_string(Inout_String,tty=.true.)
>
> C Now that we know the number of micro species,
> C allocate memory to SPCDATA_SUB type.
>
> allocate
> & (Eosdata(imat)%Incompress%Spcdata(isp)
> & %Spcdata_Sub(
> & Eosdata(imat)%Incompress%Spcdata(isp)%nsp
> & )
> & )
>
> C Loop over each micro species at a time and read the
> C necessary data. At the same time, compute the
macro
> C species gas constant and specific heat at constant
> C pressure.
>
> Eosdata(imat)%Incompress%Spcdata(isp)%rgas=zero
> Eosdata(imat)%Incompress%Spcdata(isp)%cp =zero
>
> do isp_sub=1,Eosdata(imat)%Incompress
> & %Spcdata(isp)%nsp
> if(done_nxtlne) then
> Eosdata(imat)%Incompress%Spcdata(isp)
> & %Spcdata_Sub(isp_sub)%id
> & =Eosdata(imat)%Incompress%Spcdata(isp)%id
> Eosdata(imat)%Incompress%Spcdata(isp)
> & %Spcdata_Sub(isp_sub)%massfrac
> & =one
> Eosdata(imat)%Incompress%Spcdata(isp)
> & %Spcdata_Sub(isp_sub)%molwgt
> & =one
> Eosdata(imat)%Incompress%Spcdata(isp)
> & %Spcdata_Sub(isp_sub)%gamma
> & =five3rd
> else
> call read_nxtlne(line,error1,Inbuffer)
> if(error1) error=.true.
> read(line,*) Eosdata(imat)%Incompress
> & %Spcdata(isp)
> & %Spcdata_Sub(isp_sub)%id ,
> & Eosdata(imat)%Incompress
> & %Spcdata(isp)
> & %Spcdata_Sub(isp_sub)%massfrac,
> & Eosdata(imat)%Incompress
> & %Spcdata(isp)
> & %Spcdata_Sub(isp_sub)%molwgt ,
> & Eosdata(imat)%Incompress
> & %Spcdata(isp)
> & %Spcdata_Sub(isp_sub)%gamma

```

```

> endif
> Inout_String=' '
> write(Inout_String,"( ' spc_sub =',a13 ,';',
> & ' massfrac=',f13.8,';',
> & ' molwgt =',f13.8,
> & /,' gamma =',f13.8
> & )"
> & ) Eosdata(imat)%Incompress%Spcdata(isp)
> & %Spcdata_Sub(isp_sub)%id
> & Eosdata(imat)%Incompress%Spcdata(isp)
> & %Spcdata_Sub(isp_sub)%massfrac
> & Eosdata(imat)%Incompress%Spcdata(isp)
> & %Spcdata_Sub(isp_sub)%molwgt
> & Eosdata(imat)%Incompress%Spcdata(isp)
> & %Spcdata_Sub(isp_sub)%gamma
> call write_string(Inout_String,tty=.true.)
>
> Eosdata(imat)%Incompress%Spcdata(isp)%rgas
> & = Eosdata(imat)%Incompress%Spcdata(isp)%rgas
> & +runiv*Eosdata(imat)%Incompress%Spcdata(isp)
> & %Spcdata_Sub(isp_sub)%massfrac
> & /Eosdata(imat)%Incompress%Spcdata(isp)
> & %Spcdata_Sub(isp_sub)%molwgt
>
> Eosdata(imat)%Incompress%Spcdata(isp)%cp
> & = Eosdata(imat)%Incompress%Spcdata(isp)%cp
> & +runiv*Eosdata(imat)%Incompress%Spcdata(isp)
> & %Spcdata_Sub(isp_sub)%massfrac
> & *Eosdata(imat)%Incompress%Spcdata(isp)
> & %Spcdata_Sub(isp_sub)%gamma
> & /( Eosdata(imat)%Incompress%Spcdata(isp)
> & %Spcdata_Sub(isp_sub)%molwgt
> & * (
Eosdata(imat)%Incompress%Spcdata(isp)
> & %Spcdata_Sub(isp_sub)%gamma
> & -one
> & )
> & )
>
> enddo
>
> enddo
>
> C . . . . .
.
3715a4286,4292
> c case('incompress') ! dtr
> c il=il-1 ! dtr
> c do isp=1,Eosdata(imat)%Incompress%nsp
> c il =il+1
> c Spc_Id(il)=Eosdata(imat)%Incompress
> c & %Spcdata(isp)%id
> c enddo
3875a4453,4456
> cdtr incorporated into this routine
> c call read_anl_input ! CTZ
>

```

```
>
*****
3883a4465
>
```

RES_LIN_CONT

Updates the continuity equation as a result of some calculated change in pressure.

```
77a78
>
79a81
>      use      Parallel_module_c ! DTR
93c95
<      &      cellface_tilde_change
---
>      &      cellface_tilde_change, cellface_coeff
103a106,108
>      use      anl_data_module_c ,      only:
>      &      iup_rho, ictz, matrix_osolv      ! CTZ
>
122c127,133
<      real      (kind      =real_kind )
---
>      integer      (kind      = int_kind )
>      &      :: i ,ii,k1,k2,itmp      ! CTZ
>
>      integer      (kind      = int_kind )
>      &      ::      ATMP(1)      ! DTR
>
>      real      (kind      = real_kind )
124a136,148
>      real      (kind      =real_kind ),      !      DTR
>      &      dimension      ( nconns )
>      &      :: DDPDXM,DDPDYM,DDPDZM
> CHPFD_DISTRIBUTE DDPDXM      (      _BLK_)
> CHPFD_DISTRIBUTE DDPDYM      (      _BLK_)
> CHPFD_DISTRIBUTE DDPDZM      (      _BLK_)
>
>      real      (kind      =real_kind ),      ! CTZ
>      &      dimension      (      nnodemax)
>      &      :: SCRATCHN4,      DELRHO, RESCOLD, RESCOLD
> CHPFD_DISTRIBUTE SCRATCHN4      (      _BLK_)
> CHPFD_DISTRIBUTE DELRHO      (      _BLK_)
>
136d159
<
223a247
>
260,261c284,319
<      call cellface_tilde_change(DELTA_PT,RHOOLDT,
<      &      DELTAUTM,DELTAVTM,DELTAWTM)
---
>
```

```

>
*****
>
> C      CTZ Approach
> C      -----
>
>      if(ictz.eq.1)then
>
> C      Compute changes in velocities at median mesh points (tilda
velocs)
>
>      call gradt_sm(DELTAPT,DDPDXM,DDPDYM,DDPDZM)
>
> c      call cellface_tilde_change(U,V,W,RHO,
> c      &      DELTAUT,DELTAUT,DELTAUT,RHOM,
> c      &      DELTAUTM,DELTAUTM,DELTAUTM,VOLFLOWM,
> c      &      DDPDXM,DDPDYM,DDPDZM)
>
> c      DELTAUTM=-CNUM*DDPDXM*rpgs2
> c      DELTAUTM=-CNVM*DDPDYM*rpgs2
> c      DELTAUTM=-CNWM*DDPDZM*rpgs2
>
>      DELTAUTM=-half*dt*(one/RHOT(1,:)+one/RHOT(2,:))*DDPDXM*rpgs2
>      DELTAUTM=-half*dt*(one/RHOT(1,:)+one/RHOT(2,:))*DDPDYM*rpgs2
>      DELTAUTM=-half*dt*(one/RHOT(1,:)+one/RHOT(2,:))*DDPDZM*rpgs2
> c      call cellface_tilde_change(DELTAPT,RHOOLDT,
> c      &      DELTAUTM,DELTAUTM,DELTAUTM)
>
> C      Compute change of pressure gradient at median-mesh boundary
> C      Use for the gradient change the SCRATCHM array
>
>      else      ! do the non-ctz-way
>
>      call cellface_tilde_change(DELTAPT,RHOOLDT,
>      &      DELTAUTM,DELTAUTM,DELTAUTM)
>
>      endif      ! ictz
276a335,337
> C      if(matrix_osolv)then
> C      DELTAT=SIGN(MIN(ABS(DELTAT),0.15_real_kind*ABS(T)),DELTAT)
> C      else
278c339,340
<      DELTAT=MERGE(MAX(DELTAT,zero),DELTAT,T.LE.zero)
---
> C      end if
>      DELTAT=MERGE(MAX(DELTAT,zero),DELTAT,T.LE.zero)
279a342,343
> c      print *, 'max dt = ', global_maxval(DELTAT,MASK=NODETYPE.gt.0)
> c      print *, 'min dt = ', global_minval(DELTAT,MASK=NODETYPE.gt.0)
307a372
>
308a374
>
309a376,381
>      if(matrix_osolv)then
> c      DELTARHO=MIN(MAX(DELTARHO,-0.9_real_kind*RHO),
> c      &      10.0_real_kind*RHO)

```



```

>
243a258,259
>
>      end if ! CTZ
293a310,321
>
> C   If first order upwind (iup_tmp =1) then compute DELTAHM as below
> C
>      if(iup_tmp.eq.1) then      ! CTZ
>          VFXUPW =one
>          WHERE(COURANTM.LT.zero)
>              VFXUPW = -one
>          ENDWHERE
>
>      DELTAHM=half*(DELTAHT(1,:)*(one+VFXUPW)+DELTAHT(2,:)*
>      &          (one-VFXUPW))
>      else
301c329
<
---
>      end if

```

RES_LIN_UVW

Updates the momentum equation as a result changes in velocity.

```

65a66
>
80a82,83
>      use      cellface_module_d      ,only:
>      &      cellface_tilde_change
86a90,93
>      use      nodetypes_module_c      ! CTZ
>      use      anl_data_module_c      , only:
>      &      iup_tmp, iup_uvw, ictz ! CTZ
>      use      Parallel_module_c      !dtr
100a108,110
>      integer      (kind      = int_kind )
>      &      :: i      , ipass ,      iprint      ! CTZ
>
111a122,137
>      real      (kind      =real_kind ),      ! CTZ
>      &      dimension      ( 2,nconns )
>      &      :: SCRATCHT4
>      CHPDFD_DISTRIBUTE SCRATCHT4      (_STR_,_BLK_)
>
>      real      (kind      =real_kind ),      ! CTZ
>      &      dimension      (      nnodemax)
>      &      :: SCRATCHN4
>      CHPDFD_DISTRIBUTE SCRATCHN4      (      _BLK_)
>
>
>      real      (kind      =real_kind )      ! CTZ
>      &      :: alfa      ! CTZ

```



```

>          DELTAVTM=zero
>          DELTAWTM=zero
>          ENDWHERE
>      endif
>
>      DELTAUM=half*(DELTAUT(1,:)*(one+VFXUPW)
>      &          +DELTAUT(2,:)*(one-VFXUPW))
>      DELTAVM=half*(DELTAUT(1,:)*(one+VFXUPW)
>      &          +DELTAUT(2,:)*(one-VFXUPW))
>      DELTAWM=half*(DELTAUT(1,:)*(one+VFXUPW)
>      &          +DELTAUT(2,:)*(one-VFXUPW))
>
>      C          *
>
>      C          *
>
>      C          *****
>
>      else          ! CTZ approach
>
>          DELTAUTM=half*(DELTAUT(1,:)+DELTAUT(2,:))
>          DELTAVTM=half*(DELTAUT(1,:)+DELTAUT(2,:))
>          DELTAWTM=half*(DELTAUT(1,:)+DELTAUT(2,:))
>
>      if(flagnofluxm) then
>          WHERE(NOFLUXM)
>              DELTAUTM=zero
>              DELTAVTM=zero
>              DELTAWTM=zero
>          ENDWHERE
>      endif
>
>
>      if(iup_uvw.eq.1) then          ! CTZ
>
>          DELTAUM=half*(DELTAUT(1,:)*(one+VFXUPW)
>          &          +DELTAUT(2,:)*(one-VFXUPW))
>          DELTAVM=half*(DELTAUT(1,:)*(one+VFXUPW)
>          &          +DELTAUT(2,:)*(one-VFXUPW))
>          DELTAWM=half*(DELTAUT(1,:)*(one+VFXUPW)
>          &          +DELTAUT(2,:)*(one-VFXUPW))
>      else
>
>          DELTAUM =( two*socit*dt*DELTAUM
284c378
<          DELTAVM =( two*socit*dt*DELTAVM
---
>          DELTAVM =( two*socit*dt*DELTAVM
291c385
<          DELTAWM =( two*socit*dt*DELTAWM
---
>          DELTAWM =( two*socit*dt*DELTAWM
298a393,397
>      endif          ! iup_uvw-ctz
>      endif          ! CTZ approach
>      endif          ! high mach scheme
>      endif          ! iter controls

```

```

>
299a399
>

```

SOLVE_CONT_MODULE_C

Adds arrays used for the matrix solution.

```

132a133
>
163a165
>      public  :: A0              ,A0T              ,BB              !ctz-dtr
218a221,235
>      real    (kind=real_kind    ),
>      &        dimension  ( :,      :),
>      &        allocatable
>      &        :: A0T
>      CHPFD_DISTRIBUTE A0T        (_STR_,_BLK_) !ctz-dtr
>
>      real    (kind=real_kind    ),
>      &        dimension  (      :),
>      &        allocatable
>      &        :: A0      ,  BB
>      CHPFD_DISTRIBUTE A0        (      _BLK_)
>      CHPFD_DISTRIBUTE BB        (      _BLK_) !ctz-dtr
>
>
>
C_____
-
-
>
400a418,428
>      allocate(A0T        ( 2,nconns  ),
>      &          stat=istatus
>      &          )          !ctz-dtr
>      ierror=ierror+abs(istatus)
>
>      allocate(A0        (  nnodemax  ),
>      &          BB        (  nnodemax  ),
>      &          stat=istatus
>      &          )          !ctz-dtr
>      ierror=ierror+abs(istatus)
>
408a437,438
>
>
467a498,508
>      ierror=ierror+abs(istatus)
>
>      deallocate(A0T      ,
>      &          stat=istatus
>      &          )          !ctz-dtr
>      ierror=ierror+abs(istatus)
>
>      deallocate(A0      ,

```

```

>      &          BB          ,
>      &          stat=istatus
>      &          )          !ctz-dtr

```

SOLVE_GCR_S

Adds the ability to modify the magnitude of changes in scalar equations.

```

55a56
>
56a58
>      use          timedata_module_c
66a69,71
>      use          anl_data_module_c          ,only:
>      &          slv_tol_s, use_urfp, urf_p,use_urft, urf_t,      ! DTR
>      &          use_urftke, urf_tke, use_urfeps, urf_eps      ! DTR
130c135
<      &          :: inorth          ,north
---
>      &          :: inorth          ,north, ii !DTR
189,190c194,197
<          test1=0.1_real_kind*test1
<          TEST2=0.1_real_kind*TEST2
---
>          test1=0.1_real_kind*test1*slv_tol_s
>          TEST2=0.1_real_kind*TEST2*slv_tol_s
> c          test1=0.1_real_kind*test1
> c          TEST2=0.1_real_kind*TEST2
304a312,314
>          if(equation.eq.'cont') then
> c          print *, 'cont test1'
>          end if
310a321,323
>          if(equation.eq.'cont') then
> c          print *, 'cont test2'
>          end if
318c331,355
<      &          ) then
---
>      &          ) then
>
>          if(equation.eq.'cont')then
>          if(use_urfp)then
>              QX=QX*urf_p
>          end if
>          end if
>
>          if(equation.eq.'temp')then
>          if(use_urft)then
>              QX=QX*urf_t
>          end if
>          end if
>
>          ! if(equation.eq.'tke')then

```

```

>          ! if(use_urftke)then
>          !   QX=QX*urf_tke
>          ! end if
>          ! end if
>
>          ! if(equation.eq.'eps')then
>          ! if(use_urfeps)then
>          !   QX=QX*urf_eps
>          ! end if
>          ! end if
362a400,404
> c          open(unit=55,file='gcr-icnv.dat')
> c          do ii=1,nnodemax
> c             write(55,*)ii,RESX(ii)
> c          end do
> c          close(55)
378a421
>

```

SOLVE_HYDRO

Modifications primarily relating to the ANL reformulated continuity equation.

```

179c179
< C          MASSFRAC      = Macro species' mass fractions at time-level n.
---
> C          MASSFRAC      = Macro species mass fractions at time-level n.
679a680
>
688a690,691
>          use          anl_data_module_c          ,only:
>          &            itmaxout, ictz, matrix_solv, plot_res
734a738,739
>          use          solve_hydro_module_c      ,only:      ! DTR
>          &            RESUL,RESVL,RESWL,RESTL,RESCL
754c759
<          &            :: baddata
---
>          &            :: baddata,dtrlog
760c765,766
<          &            npmin          ,nrhomin          ,ntmin
---
>          &            npmin          ,nrhomin          ,ntmin,
>          &            k1,k2,ii !dtr
768d773
<
778a784,793
> c          real          (kind          =real_kind  ),      !CDTR
> c          &            dimension      ( nconns)
> c          &            :: DPDXM,DPDYM,DPDZM
>
>
>          real          (kind          =real_kind  ),      !CDTR
>          &            dimension      (2, nconns)

```

```

>      &      :: TMPT1, TMPT2, TMPT3
>
>
814a830
>
821a838,840
>      if(ictz.eq.1)then
>          call zero_walls(.true.)
>      end if
906c925
< C          density] because they won't be exactly consistent with
the
---
> C          density] because they will not be exactly consistent with
the
1230c1249,1253
<
---
1256c1279
< C          wall.  But we don't care about this because
---
> C          wall.  But we do not care about this
because
1331a1355,1357
>          if(ictz.eq.1) then
>          P=POLD      !CTZ-DTR
>      end if
1335a1362,1366
>          if(ictz.eq.1)then
>          U=UOLD      !CTZ
>          V=VOLD      !CTZ
>          W=WOLD      !CTZ
>      end if
1343d1373
<
1375,1376c1405
<      call gatherh_vn(U,V,W,
<      & UT,VT,WT)
---
>      call gatherh_vn(U,V,W,UT,VT,WT)
1404,1405c1433,1434
<      call gradbcm_sn(P,P_OPEN,SCRATCHM1,
<      & DPDX,DPDY,DPDZ)
---
>
>      call gradbcm_sn(P,P_OPEN,SCRATCHM1,DPDX,DPDY,DPDZ)
1437c1466
<          SCRATCHN1=DPDX
---
>          SCRATCHN1=DPDX
1496,1497c1525
<      call gatherh_vn(RESU,RESV,RESW,
<      & SCRATCHT1,SCRATCHT2,SCRATCHT3)
---
>      call gatherh_vn(RESU,RESV,RESW,SCRATCHT1,SCRATCHT2,SCRATCHT3)
1499,1510c1527,1563
<      if(tilde_scheme.eq.'highmach') then

```

```

<         call cellface_tilde(P,PT,
<     &     RHOOLD,RHOOLDT,SCRATCHT1,SCRATCHT2,SCRATCHT3,
<     &     UTILDEM,VTILDEM,WTILDEM,VOLFLOWM)
<     else
<         call cellface_tilde(P,U,V,W,
<     &     PT,UT,VT,WT,
<     &     SCRATCHN1,SCRATCHN2,SCRATCHN3,
<     &     POLD,RHOOLD,
<     &     POLDT,RHOOLDT,SOUNDOLDT,SCRATCHT1,SCRATCHT2,SCRATCHT3,
<     &     UTILDEM,VTILDEM,WTILDEM,VOLFLOWM)
<     endif
---
> C     Compute guess VOLFLOWM needed at cycle one for the computation
> C     of "tilde" velocity wiith CTZ approach
> C     Initialize XMASSV to zero
>
> CDTR This is step #1 of the reformulated continuity equation in my
notes
> CDTR Quick and dirty check to reinit VOLFLOWM on a restart
>     if(ictz.eq.1)then      !cdtr
>         if(
>     &         (ncyc.eq.1)
>     &         .or.
>     &         ((Global_Sum(VOLFLOWM).eq.0).and.restart)
>     &         ) then      ! CTZ
>         VOLFLOWM=half*
>     &         (
>     &         (UT(1,:)+UT(2,:))*AXM+
>     &         (VT(1,:)+VT(2,:))*AYM+
>     &         (WT(1,:)+WT(2,:))*AZM
>     &         )
>     else
>         VOLFLOWM = VOLFLOLDM
>     endif
> end if ! cdtr
>
>     if(ictz.ne.1) then    ! ctz actually commented the below out
>     if(tilde_scheme.eq.'highmach') then
>         call cellface_tilde(P,PT,
>     &         RHOOLD,RHOOLDT,SCRATCHT1,SCRATCHT2,SCRATCHT3,
>     &         UTILDEM,VTILDEM,WTILDEM,VOLFLOWM)
>     else
>         call cellface_tilde(P,U,V,W,
>     &         PT,UT,VT,WT,
>     &         SCRATCHN1,SCRATCHN2,SCRATCHN3,
>     &         POLD,RHOOLD,
>     &         POLDT,RHOOLDT,SOUNDOLDT,SCRATCHT1,SCRATCHT2,SCRATCHT3,
>     &         UTILDEM,VTILDEM,WTILDEM,VOLFLOWM)
>     endif
>     end if !ictz=0
1624d1676
< C     Start the outer-iteration loop.
1625a1678,1682
> C     Start the outer-iteration loop.
> c     if(matrix_solv) then
> c         iterout_use=7
> c         limit_semiimplicit=14

```



```

> c      else
1631a1689,1690
> cdtr not commenting this out because not sure why CTZ put this in
> cdtr      iterout_use=itmaxout      !      CTZ
1632a1692
> c      end if ! matrix_solv
1641a1702,1717
> cdtr      this is step two corresponding to ctz approach
>
>          if(ictz.eq.1) then
>              call cellface_tilde(
>                  &      PT,UT,VT,WT,UTILDEM,VTILDEM,WTILDEM,VOLFLOWM)
>              end if !ictz
>
>
*****
>
>
1737c1813,1814
<      &      Global_MINVAL(      RHO,MASK=NODETYPE.GE.0).LE.zero.or.
---
>      &      Global_MINVAL(      RHO,MASK=NODETYPE.GE.0).LE.zero
>      &      .or.
1744d1820
<
1925a2002,2003
>
>
2009a2088
>
2027a2107,2115
2032a2121
>
2036a2136,2139
2281a2410,2416
> C      Save the Volumetric flow rate at median mesh to be used as a
> C      first guess at the next cycle in cellface_tilde      ! CTZ
>
> c      print *, 'outer iteration has converged, saving vflowm'
>          if (ictz.eq.1) then ! ctz
>              VOLFLOLDM = VOLFLOWM ! ctz
>          end if      ! ctz
2315a2451
>
2317a2454,2457
>          if(ictz.eq.1)then
>              DPDT      = 0.0      ! CTZ
>          end if      !ictz
>
2323d2462
<
2386d2524
<
2388a2527
>      &      VFXUPW,
2441a2581,2586

```

C

```

>
*****
>
>      if(plot_res)then
> c      call write_residuals
>      end if
>

```

C

SOLVE_OR_RES_NL_CONT

Modifications arising from the reformulated continuity equation, residual monitoring and convergence control, and the matrix solver.

```

153a154
>
157c158,159
<      &          global_minval
---
>      &          global_minval,global_sum          ,
>      &          global_all,scalar_sum,parallel_collate,PEInfo! DTR
161,162c163,164
<      use          eosdata_module_c          ,only:
<      &          nmats
---
>      use          eosdata_module_c          ! ,only:
> c      &          nmats
164a167,168
>      use          cellface_module_d          ,only:
>      &          cellface_coeff
167c171,172
<      &          ntp_free          ,ntp_free_src
---
>      &          ntp_free          ,ntp_free_src,
>      &          ntp_freeslip, ntp_intrfc
191a197,201
>      use          anl_data_module_c          ,only:
>      &          ictz, iup_rho, iup_tmp, iup_uvw,
>      &          matrix_solv, imatrixbug,matrix_osolv,
>      &          norm_fact_p, r_red_fact_p
>      use          eosdata_module_c          ! DTR
210c220
<      &          :: ireg          ,iterin_mn          ,iterin_mx
---
>      &          :: ireg          ,iterin_mn          ,iterin_mx,nnodemaxtot
214a225,269
> CDTR-----
--
>      real          (kind          =real_kind )
>      &          :: resmax, test1, maxvar,tmp1,tmp2,tmp3,tmp4,tmp5
>
>      integer       (kind          =int_kind )
>      &          :: iptc_its,iptc_mx_its          ! # iterations ptc requires
>      &          ,ii, k1, k2,jj          ! upwinding flag, loop
counter,indices

```

```

>      &          ,iksp_reason, iter_dtr
>
>      real      (kind      =real_kind ),
>      &          dimension  (2, nconns)
>      &          :: DRESUT, DRESVT, DRESWT,AT,
>      &          DRESRT1,DRESRT2,DRESRT3
> CHPFD_DISTRIBUTE DRESUT      (      _BLK_ )
> CHPFD_DISTRIBUTE DRESVT      (      _BLK_ )
> CHPFD_DISTRIBUTE DRESWT      (      _BLK_ )
> CHPFD_DISTRIBUTE DRESRT1     (      _BLK_ )
> CHPFD_DISTRIBUTE DRESRT2     (      _BLK_ )
> CHPFD_DISTRIBUTE DRESRT3     (      _BLK_ )
>
>
>      real      (kind      =real_kind ),
>      &          dimension  (nnodemax)
>      &          :: DRESU, DRESV, DRESW,
>      &          DRESR1,DRESR2,DRESR3
>      &          ,TEST2
> CHPFD_DISTRIBUTE DRESU      (      _BLK_ )
> CHPFD_DISTRIBUTE DRESV      (      _BLK_ )
> CHPFD_DISTRIBUTE DRESW      (      _BLK_ )
> CHPFD_DISTRIBUTE DRESR1     (      _BLK_ )
> CHPFD_DISTRIBUTE DRESR2     (      _BLK_ )
> CHPFD_DISTRIBUTE DRESR3     (      _BLK_ )
> CHPFD_DISTRIBUTE TEST2      (      _BLK_ )
>
>      real      (kind      =real_kind ),
>      &          dimension  (nnodemax)
>      &          :: CDRHODP
> CHPFD_DISTRIBUTE CDRHODP      (      _BLK_ )
>
>      logical   (kind      =log_kind  )
>      &          :: convin_test1,convin_test2
>
>
>  CDTR-----
--
>
246,250d300
< C      Initialize some quantities.
<
<      rpgs2      =one/pgs**2
<
<
C
*****
259a310,331
>      if(ictz.eq.1)then
>
>  C          CTZ      modified   it:   QFLOWOUT   is   =0.0   where
P_OPEN=hugenum(interior)
>  C
>          SCRATCHN1= (U-UMESH)*AXN_OPEN
>      &          + (V-VMESH)*AYN_OPEN
>      &          + (W-WMESH)*AZN_OPEN
>          WHERE (P_OPEN .GE.hugenum .OR.
>  C      &          SCRATCHN1.GE.zero .OR.      !      CTZ

```

```

> C      &      NODETYPE .EQ.ntp_free      .OR.      !      CTZ
>      &      NODETYPE .EQ.ntp_freeslip.OR.      !      CTZ      Temporary
>      &      NODETYPE .EQ.ntp_free_src.OR.
>      &      NODETYPE .EQ.ntp_intrfc
>      &      )
>      QFLOWOUT =zero
>      RHOOUTFLOW=zero
>      ELSEWHERE
>      QFLOWOUT =SCRATCHN1
>      RHOOUTFLOW=RHO
>      ENDWHERE
>
>      else
261c333
<      SCRATCHN1= (U-UMESH)*AXN_OPEN
---
>      SCRATCHN1= (U-UMESH)*AXN_OPEN
264c336
<      WHERE (P_OPEN      .GE.hugenum      .OR.
---
>      WHERE (P_OPEN      .GE.hugenum      .OR.
270,276c342,348
<      QFLOWOUT =zero
<      RHOOUTFLOW=zero
<      ELSEWHERE
<      QFLOWOUT =SCRATCHN1
<      RHOOUTFLOW=RHO
<      ENDWHERE
<
---
>      QFLOWOUT =zero
>      RHOOUTFLOW=zero
>      ELSEWHERE
>      QFLOWOUT =SCRATCHN1
>      RHOOUTFLOW=RHO
>      ENDWHERE
>      end if !CTZ
355c427
< C      Density-based scheme.
---
> C      Initialize some quantities.
356a429,434
>      rpgs2      =one/pgs**2
>
>
*****
>
> CDTR      Clear out the BB vector.
>      BB = zero
358d435
<
C
-----
359a437,540
> CDTR      Begin Tzanos derivation of matrix coefficients
> CDTR      *
> CDTR      *
> CDTR      Compute the portion of the derivative of the residual with

```



```

>         else
> c         RDRESCDP=DRHODPT+DRHODTP*CBIG
>         RDRESCDP=DRHODPT+DRHODTP*CBIG
>         end if
600a891,893
> c         do ii=1,nnodemax
> c             print *, 'psource ', ii, PSOURCE(ii)
> c         end do
670d962
<
673d964
<
724a1016,1024
> c         the matrix terms are not symmetricized so dont symmetricize
these
>         if ((matrix_solv).or.(matrix_osolv)) then
>             BB = -RESCL
>         end if                                ! end matrix_solv (BB)
>
>         if(matrix_osolv)then
>
>         end if
>
730c1030,1031
<         endif
---
>         endi             f
>
755d1055
<
767a1068,1087
>
> CDTR-----
--
>         if(matrix_solv) then                !start dtr-matrix-solve
>
>         nnodemaxtot=Scalar_Sum(nnodemax)
>
> solve_petsc_cont_par('cont',icon_cont,cont_it,iterin_mn,
> &             iterin_mx,convin,test1_cont,DELTAPMIN,
> &             DELTAPMAX,TEST2_CONT,nnodemaxtot)
>
>         elseif(matrix_osolv.or.matrix_solv) then ! else_matrix_solve
>
> c         will see how orig chad performs with petsc.  Coding is
different.
>         nnodemaxtot=Scalar_Sum(nnodemax)
>
>
> solve_petsc_cont_par_2('cont',icon_cont,cont_it,iterin_mn,
> &             iterin_mx,convin,test1_cont,DELTAPMIN,
> &             DELTAPMAX,TEST2_CONT,nnodemaxtot)
>         else
> CDTR-----
--
>
768a1089

```

```

>
772a1094,1095
>         end if ! end_matrix_solve
>
773a1097
>
813d1136
<
830d1152
<
834d1155
<
852a1181,1247
>         if(iterout.le.19)then
> C             SPAREN(1,:)=DELTAP
> C             SPAREN(3,:)=DELTAV
> C             SPAREN(4,:)=RESC
> C             SPAREN(5,:)=RESCL
> c             SPAREN(6,:)=CONTAB(1,:)
> c             SPAREN(7,:)=CONTAB(4,:)
> c             SPAREN(8,:)=CONTAB(6,:)
> c             SPAREN(9,:)=NODETYPE
>         end if
>
<         SCRATCHM3=two/(RHOOLDT(1,:)+RHOOLDT(2,:))
<         HM         =HM+half*SCRATCHM3*(DELTAPT(1,:)+DELTAPT(2,:))
---
>
>         if(iup_tmp.eq.1) then          ! CTZ
>             VFXUPW =one
>             WHERE(COURANTM.LT.zero)
>                 VFXUPW = -one
>             ENDWHERE
>
>             HM=half*(HT(1,:)*(one+ VFXUPW)+ HT(2,:)*(one-VFXUPW))
>
>         else          ! CTZ
>
>             SCRATCHM3=two/(RHOOLDT(1,:)+RHOOLDT(2,:))
>             HM=HM+half*SCRATCHM3*(DELTAPT(1,:)+DELTAPT(2,:))
943a1350
>         end if ! iup_tmp ! CTZ
953a1361
>
986a1395,1404
> C     If first order upwind (iup_uvw =1) then compute UM etc as below
>
>         if(iup_uvw.eq.1) then          ! CTZ
>
>             UM =half*(UT(1,:)*(one+ VFXUPW) + UT(2,:)*(one-VFXUPW))
>             VM =half*(VT(1,:)*(one+ VFXUPW) + VT(2,:)*(one-VFXUPW))
>             WM =half*(WT(1,:)*(one+ VFXUPW) + WT(2,:)*(one-VFXUPW))
>
>         else          ! CTZ
>
1008a1427
>         end if ! CTZ

```



```

>      use      anl_data_module_c      ,only:
>      &      norm_fact_t,
>      &      r_red_fact_t, i_no_e
344a350,360
> c      COEFFDT=-DTOMASS*( VOL*(+DIVKGTN
> c      &
> c      &      +SRCENTH
> c      &      +RHOOUTFLOW*QFLOWOUT*(HOUTFLOW-HOLD)
> c      &      -HTC*(T-TWALL)+FRICC*( (U-UWALL)**2
> c      &      + (V-VWALL)**2
> c      &      + (W-WWALL)**2
> c      &      )
> c      &      -RESTL
> c      &      )
>
642a659
>      if(i_no_e.ne.1)then
646a664,666
>      else
>      icon_tmp=1
>      end if
648d667
<
686a706,707
>
>
718a740,752
>
>      if(ncyc.lt.11)then
>
>      norm_fact_t=max(norm_fact_t,(global_maxval(RESTL,
>      &      MASK=NODETYPE.gt.0)))
>      end if
>
>      r_red_fact_t=global_sum(abs(RESTL),MASK=NODETYPE.gt.0)
>      &      /norm_fact_t
>
>
>
*****
>

```

C

SOLVE_OR_RES_NL_UVW

Modifications arising from residual monitoring and convergence control.

```

187a188
>
191a193
>      &      ,global_sum
200,201c202
<      use      nodetypes_module_c      ,only:
<      &      ntp_free      ,ntp_free_src

```

```

---
> use nodetypes_module_c
223a225,234
> use anl_data_module_c ,only:
> & iup_uvw,ictz, ! CTZ
> & norm_fact_u, ! DTR
> & norm_fact_v, ! DTR
> & norm_fact_w, ! DTR
> & r_red_fact_u, ! DTR
> & r_red_fact_v, ! DTR
> & r_red_fact_w, ! DTR
> & ,gravx,gravy,gravz ! CTZ/DTR
> & ,rhozero, vexpansion ! DTR
242c253
< & :: ireg ,iterin_mn ,iterin_mx
---
> & :: ireg ,iterin_mn ,iterin_mx, ii,k1,k2
245,246c256,257
< & :: deltauvw_glo,testmin ,testtmp
<
---
> & :: deltauvw_glo,testmin ,testtmp,resnorm
> & ,tmp
307a319,338
> if(ictz.eq.1)then
> DTOMASS =dt/MAX(RHO*VOL,tinyum)
>
> SCRATCHN1= (U-UMESH)*AXN_OPEN
> & + (V-VMESH)*AYN_OPEN
> & + (W-WMESH)*AZN_OPEN
> WHERE(P_OPEN .GE.hugenum .OR.
> C & SCRATCHN1.GE.zero .OR. ! CTZ
> C & NODETYPE .EQ.ntp_free .OR. ! CTZ
> & NODETYPE .EQ.ntp_freeslip.OR. ! CTZ Temporary
> & NODETYPE .EQ.ntp_free_src.OR.
> & NODETYPE .EQ.ntp_intrfc
> & )
> COEFFDUVW =one
> RHOOUTFLOW=zero
> ELSEWHERE
> COEFFDUVW =one-DTOMASS*RHO*SCRATCHN1
> RHOOUTFLOW=RHO
> ENDWHERE
> else
325c356
<
---
> end if
344d374
<
382a413
>
385a417,421
>
> c computer min wall temperature
> c tmp = global_minval(T,MASK=NODETYPE.gt.0)
> tmp=308.

```

```

>
386a423
> c      &      -dt*(RHO*gravx)/MAX(RHO,tinynum) ! buoyancy term CTZ
391a429,430
> c      &      +dt*(RHO-rhozero)*gravx/MAX(RHO,tinynum) ! buoyancy
term CTZ
>
392a432,433
> c      &      -dt*(SCRATCHN1*gravity)/MAX(RHO,tinynum) ! buoyancy term
CTZ
>      &      -dt*(gravity*vexpansion*(T-tmp))
397a439,443
> c      &      +dt*(RHO-rhozero)*gravity/MAX(RHO,tinynum) ! buoyancy
term CTZ
>
>
>      SCRATCHN1=zero
>
398a445
> c      &      -dt*(RHO*gravz)/MAX(RHO,tinynum) ! buoyancy term CTZ
404c451
<
---
> c      &      +dt*(RHO-rhozero)*gravz/MAX(RHO,tinynum) ! buoyancy
term CTZ
523a571
>
648a697,698
> c      print *, 'solve_or_res_nl_uvw iterin_mx changed'
> c      iterin_mx=iterin_mx*5
688d737
<
697a747
>
700a751
>
702,704c753,769
<      UM      =UM      +DELTAUM
<      VM      =VM      +DELTAUM
<      WM      =WM      +DELTAUM
---
>
>
>      if(iup_uvw.eq.1) then      ! CTZ
>
>      VFXUPW =one
>      WHERE(COURANTM.LT.zero)
>      VFXUPW = -one
>      ENDWHERE
>
>      UM = half*(UT(1,:)*(one+ VFXUPW)+ UT(2,:)*(one-VFXUPW))
>      VM = half*(VT(1,:)*(one+ VFXUPW)+ VT(2,:)*(one-VFXUPW))
>      WM = half*(WT(1,:)*(one+ VFXUPW)+ WT(2,:)*(one-VFXUPW))
>      else      ! CTZ
>      UM      =UM      +DELTAUM
>      VM      =VM      +DELTAUM
>      WM      =WM      +DELTAUM

```

```

>         endif          ! CTZ
720a786,806
>
>
>*****C
>
>         if(ncyc.lt.11)then
>
>             norm_fact_u=max(norm_fact_u,(global_maxval(RESUL,
> & MASK=NODETYPE.gt.0)))
>             norm_fact_v=max(norm_fact_v,(global_maxval(RESVL,
> & MASK=NODETYPE.gt.0)))
>             norm_fact_w=max(norm_fact_w,(global_maxval(RESWL,
> & MASK=NODETYPE.gt.0)))
>         end if
>
>         r_red_fact_u=global_sum(abs(RESUL),MASK=NODETYPE.gt.0)
> & /norm_fact_u
>
>         r_red_fact_v=global_sum(abs(RESVL),MASK=NODETYPE.gt.0)
> & /norm_fact_v
>
>         r_red_fact_w=global_sum(abs(RESWL),MASK=NODETYPE.gt.0)
> & /norm_fact_w

```

SOLVE_PESTSC_CONT

```

*DK solve_petsc_cont
      subroutine solve_petsc_cont(equation,iconvout,
& iterin,iterin_mn,iterin_mx,
& convin,test1,
& DP,DPMIN,DPMAX,TEST2)

C#####
C#

C      Purpose:

C          Solves a given scalar equation using PETSc

C      Input Variables:

C          equation      = The name of equation being solved.
C          iterin_mn      = Minimum number of inner iterations allowed.
C          iterin_mx      = Maximum number of inner iterations allowed.
C          test1          = A scalar number for checking for convergence.
C                          If the change in quantity being iterated on is
C                          less than or equal to this value during two
C                          successive iterations, the solution is
considered
C                          to be converged.
C          DPMIN          = Minimum allowable value of the scalar.
C          DPMAX          = Maximum allowable value of the scalar.
C          TEST2          = An array for checking for convergence.  When

```

```

C          every residual in every element is less than or
C          equal to TEST2, the solution is considered to be
C          converged.

C      Output Variables:

C          iconvout      = An integer flag for nonlinear (outer-iteration)
C                        convergence. A positive value of iconvout
C                        implies full nonlinear convergence, a zero
C                        value implies full linear convergence, and a
C                        negative value implies an impossible situation
C                        of achieving even linearized convergence at the
C                        current time step.
C          iterin        = Number of inner iterations required to converge.
C          convin        = A logical flag indicating if the solution
C                        got fully converged within the required number
C                        of iterations.
C          DP            = Converged value of the scalar.

C#####
C#

C      Module list and other preliminaries.

#include "macros.HH"

        use      Kinds_module_c
        use      Parallel_module_c
        use      constants_module_c
        use      dimensions_module_c
        use      arrays_glo_module_c      , only:
&      NODEST
        use      inout_module_d      , only:
&      blank_line , Inout_String, write_string
        use      inoutdata_module_c      , only:
&      verbosity
        use      solve_cont_module_c
        use      hydro_module_c
        use      gather_module_d
        implicit none

#include "/home/cluster3/drockken/petsc-2.1.6/include/finclude/petsc.h"
#include      "/home/cluster3/drockken/petsc-
2.1.6/include/finclude/petscvec.h"
#include      "/home/cluster3/drockken/petsc-
2.1.6/include/finclude/petscmat.h"
#include      "/home/cluster3/drockken/petsc-
2.1.6/include/finclude/petscsles.h"
#include      "/home/cluster3/drockken/petsc-
2.1.6/include/finclude/petscviewer.h"
#include      "/home/cluster3/drockken/petsc-
2.1.6/include/finclude/petscksp.h"

C#####
C#

```


C Other global and local variables.

C

—

C Global variables being passed through the argument list.

```

    logical    (kind      =log_kind  ),
&              intent     (out        )
&              :: convin

    integer    (kind      =int_kind  ),
&              intent     (in         )
&              :: iterin_mn ,iterin_mx

    integer    (kind      =int_kind  ),
&              intent     (out        )
&              :: iconvout ,iterin

    real       (kind      =real_kind  ),
&              intent     (inout      )
&              :: test1

    character   (len       =*         ),
&              intent     (in         )
&              :: equation

    real       (kind      =real_kind  ),
&              dimension   (  nnodemax),
&              intent     (in         )
&              :: DPMIN   ,DPMAX
CHPFD_DISTRIBUTE DPMIN    (      _BLK_)
CHPFD_DISTRIBUTE DPMAX    (      _BLK_)

    real       (kind      =real_kind  ),
&              dimension   (  nnodemax),
&              intent     (inout      )
&              :: DP      ,
&              TEST2
CHPFD_DISTRIBUTE DP      (      _BLK_)
CHPFD_DISTRIBUTE TEST2   (      _BLK_)
```

C

—

C Local variables.

```

    logical    (kind      =log_kind  )
&              :: convin_test1,convin_test2

    integer    (kind      =int_kind  )
&              :: inorth   ,ii,jj,its,its_mx,iksp_reason,
&              i,j,k1,k2,IERR, inz,onz,dnz

    real       (kind      =real_kind  )
&              :: resmax
```

```

        integer      (kind      =int_kind  ),
        &            dimension  (nnodemax)
        &            :: TMP_LOC
CHPFD_DISTRIBUTE TMP_LOC      (_STR_,_BLK_)

C_____
-

c      KSP ksp
c      KSPConvergedReason reason

      Mat PTC_AA ! , PTC_PC
      PetscOffset I_P
c      PetscReal rel_tol
      PetscScalar P_ARRAY(1)
      PetscScalar TMP
      PetscViewer Viewer
      SLES sles
      Vec PTC_BB
      Vec PTC_P
      Vec PTC_RES
C_____
-

C#####
#

C
*****

C      Initialize some data.

      iterin      =0
      convin_test1=.false.
      convin_test2=.false.
      DP          =zero

C
*****

C      Iteratively solve for the solution by successively guessing the
C      new values and computing residuals corresponding the the new
C      values.

C      create the unknown and right hand side vectors

      call VecCreate(PETSC_COMM_WORLD, PTC_P, IERR)
      call VecSetSizes(PTC_P,PETSC_DECIDE,nnodemax,IERR)
      call VecSetFromOptions(PTC_P, IERR)

```

```

call VecDuplicate(PTC_P, PTC_BB, IERR)
call VecDuplicate(PTC_P, PTC_RES, IERR)

inz = 6
dnz = 7
onz = 6

call MatCreateSeqAIJ(PETSC_COMM_SELF, nnodemax, nnodemax,
&                    inz, PETSC_NULL_INTEGER, PTC_AA, IERR)

call MatSetFromOptions(PTC_AA, IERR)
call SLESCreate(PETSC_COMM_WORLD, sles, IERR)

c    set up work array
c    do ii=1,nnodemax
c        TMP_LOC(II)=ii-1
c    end do

do

    if(iterin.ge.iterin_mx.or.convin_test1.or.convin_test2) then
        exit
    endif

    if (iterin.ne.1) then
        call matrix_coeffs      ! matrix coeffs already set from outer
iteration
    end if

    iterin=iterin+1

C

```

```

C    Tighten the convergence criterion if the iteration count
C    exceeds one. That is, if we find that the current solution
C    will not converge the outer iteration, we might as well
C    converge well for the current outer-iteration cycle. This
C    approach speeds up the outer-iteration convergence.

    if(iterin.eq.2) then
        test1=0.1_real_kind*test1
        TEST2=0.1_real_kind*TEST2
    endif

C

```

```

C    Save old residuals and estimate the new solution values.

    if(iterin.eq.1) then

C        *** Note that for the first iteration, setting the
C        *** residual to its current outer-iteration value is
C        *** appropriate only if DP (remember, this is a

```

```

C          *** change from its current outer-iteration value)
C          *** is zero. That is why we set DP to zero at the
C          *** start of this routine.

      else

          A0=A0*RDRESCDP
          BB=BB*RDRESCDP

C      set the right hand side vector
      do I=1,nnodemax
          TMP = BB(I)
          call VecSetValue(PTC_BB, I-1, TMP, INSERT_VALUES,IERR) ! Vec
set value does not return
      end do

C      call VecSetValues(PTC_BB,nnodemax,TMP_LOC,BB,IERR)

C      finish construction. This step is needed with VecSetValue(s)

          call VecAssemblyBegin(PTC_BB,IERR)
          call VecAssemblyEnd(PTC_BB,IERR)
C_____
-

C      create the matrix

C      load the matrix diagonal coefficients

      do I=1,nnodemax
          TMP = A0(I)
          call MatSetValue(PTC_AA, I-1, I-1, TMP, INSERT_VALUES, IERR)
      enddo

C      load off diagonal coefficients

      do J=1,nconns
          k1 = NODEST(1,J)
          k2 = NODEST(2,J)
          TMP = A0T(1,J)*RDRESCDP(k1)
          print *, 'setting ', k1, k2
          call MatSetValue(PTC_AA, k1-1, k2-1, TMP, INSERT_VALUES,
IERR)

          TMP = A0T(2,J)*RDRESCDP(k2)
          call MatSetValue(PTC_AA, k2-1, k1-1, TMP, INSERT_VALUES,
IERR)
      enddo

C      finalize assembly of matrix
      print *, 'setting options'

          call MatSetOption(PTC_AA, MAT_NO_NEW_NONZERO_LOCATIONS, IERR)

          print *, 'assembling'
          call MatAssemblyBegin(PTC_AA, MAT_FINAL_ASSEMBLY, IERR)

```

```

        call MatAssemblyEnd(PTC_AA, MAT_FINAL_ASSEMBLY, IERR)

c      print out the matrix

c      call PetscViewerASCIIOpen(PETSC_COMM_WORLD,"matrix.mat",
c      &Viewer,IERR)

c      call PetscViewerSetFormat(Viewer, PETSC_VIEWER_ASCII_MATLAB
c      &, IERR)

c      call MatView(PTC_AA, viewer,IERR)
c      call PetscViewerDestroy(viewer,IERR)

c      call PetscViewerASCIIOpen(PETSC_COMM_WORLD,"matrix.default",
c      &Viewer,IERR)
c      call PetscViewerSetFormat(Viewer, PETSC_VIEWER_ASCII_DEFAULT
c      &, IERR)
c      call MatView(PTC_AA, viewer,IERR)
c      call PetscViewerDestroy(viewer,IERR)
c      call SLESSetOperators(sles, PTC_AA, PTC_AA,
&      SAME_NONZERO_PATTERN, IERR)

c      call SLESSetOperators(sles, PTC_AA, PTC_AA,
c      &      SAME_PRECONDITIONER, IERR)

c      call SLESGetKSP(sles,ksp,IERR)

c      its_mx = 50

c      call KSPSetTolerances(ksp,PETSC_DEFAULT_DOUBLE_PRECISION
c      &      ,PETSC_DEFAULT_DOUBLE_PRECISION,PETSC_DEFAULT_DOUBLE_PRECISION,
c      &      its_mx,IERR)

c      call SLESSetFromOptions(sles, IERR)

C_____
-

C      solve the system

c      call SLESSolve(sles, PTC_BB, PTC_P, ITS, IERR)

c      call KSPGetConvergedReason(ksp,reason,IERR)
c      iksp_reason = reason

c      update the solution variables

c      call VecGetArray(PTC_P, P_ARRAY, I_P, IERR)

c      first array entry located at (I_P+1) per PETSC manuals

c      do I=1,nnodemax
c      DP(I)=P_ARRAY(I_P+I) + DP(I)
c      enddo
c      call VecRestoreArray(PTC_P, P_ARRAY, I_P, IERR)

```

```

C
-

c      cleanup

c          call matrix_solve(nnodemax,nconns,NODEST,
c      &      A0,A0T,BB,RDRESCDP,DP,its,its_mx,iksp_reason
c      &      )
c          stop
c          call res_lin_cont(.false.)
c          stop
c      endif

C

```

```

C      Test for convergence.

      if(iterin.ge.max(iterin_mn,3)          .and.
&      Global_ALL(ABS(DP).LE.test1)
&      ) then
      convin_test1=.true.
      endif

      if(iterin.ge.iterin_mn          .and.
&      Global_ALL(ABS(RESC).LE.TEST2)
&      ) then
      convin_test2=.true.
      endif

      if(iterin.gt.1          .and.
&      (iterin.ge.iterin_mx.or.
&      convin_test1          .or.
&      convin_test2
&      )
&      ) then

      DP=MIN(MAX(DP,DPMIN),DPMAX)

      call res_lin_cont(.true.)

      endif
C      *** We need to call the residual routine for the last
C      *** time to assure that all node, vertex, and boundary
C      *** data for all variables are consistent at convergence,
C      *** in case the solution exceeds the imposed limits.
C      *** Obviously, it is not necessary to do so if the
C      *** outer iteration is converged (for the solver under
C      *** consideration) with the current outer-iteration
C      *** state, because we will not change anything from the
C      *** current outer-iteration values.

C

```

```

        enddo

        call VecDestroy(PTC_P, IERR)
        call VecDestroy(PTC_BB, IERR)
        call VecDestroy(PTC_RES, IERR)
        call MatDestroy(PTC_AA, IERR)
        call SLESDestroy(sles, IERR)

C      stop
C
*****

C      Set the outgoing convergence flags/variables and print necessary
C      messages.

        if(convin_test1.or.convin_test2) then
            convin=.true.
        endif

        if(iterin.eq.1) then
            iconvout=1
        else
            iconvout=0
        endif

        if(verbosity.ge.1) then
            if(convin_test1.and..not.convin_test2) then
                Inout_String=' '
                write(Inout_String,
&                  "(/, ' *****Matrix looks non-positive-definite'
&                  ' (equation=',a,').*****'
&                  /,a
&                  )"
&                  ) trim(equation),blank_line
                call write_string(Inout_String,tty=.true.)
            endif
        endif

C
*****

        end

```

SOLVE_PETSC_CONT_PAR

Solves the continuity equation using PETSc. This was originally designed for use with the reformulated continuity equation but was later abandoned in favor of using the existing continuity equation. That is solved using SOLVE_PETSC_CONT_PAR_2. This routine may be deprecated as a result.

```

*DK solve_petsc_cont_par
    subroutine solve_petsc_cont_par(equation, iconvout,
& iterin, iterin_mn, iterin_mx,
& convin, test1, DPMIN, DPMAX, TEST2, nnodemaxtot)

```

```

C#####
#

C      Purpose:

C          Solves a given scalar equation using PETSc

C      Input Variables:

C          equation      = The name of equation being solved.
C          iterin_mn     = Minimum number of inner iterations allowed.
C          iterin_mx     = Maximum number of inner iterations allowed.
C          test1         = A scalar number for checking for convergence.
C                        If the change in quantity being iterated on is
C                        less than or equal to this value during two
C                        successive iterations, the solution is
considered
C                        to be converged.
C          DPMIN         = Minimum allowable value of the scalar.
C          DPMAX         = Maximum allowable value of the scalar.
C          TEST2         = An array for checking for convergence. When
C                        every residual in every element is less than or
C                        equal to TEST2, the solution is considered to be
C                        converged.

C      Output Variables:

C          iconvout      = An integer flag for nonlinear (outer-iteration)
C                        convergence. A positive value of iconvout
C                        implies full nonlinear convergence, a zero
C                        value implies full linear convergence, and a
C                        negative value implies an impossible situation
C                        of achieving even linearized convergence at the
C                        current time step.
C          iterin        = Number of inner iterations required to converge.
C          convin        = A logical flag indicating if the solution
C                        got fully converged within the required number
C                        of iterations.

C#####
#

C      Module list and other preliminaries.

#include "macros.HH"

      use      Kinds_module_c
      use      Parallel_module_c
      use      constants_module_c
      use      dimensions_module_c
      use      arrays_glo_module_c
      use      hydro_module_c
      use      options_module_c          ,only:
&      flowtype
      use      physdata_module_c

```



```

        use      physics_module_c
        use      grads_module_d
        use      anl_data_module_c ,      only:
&      ictz, matrix_solv      ! CTZ

        use      inout_module_d      ,only:
&      blank_line ,Inout_String,write_string,open_file,
&      close_file
        use      inoutdata_module_c      ,only:
&      verbosity
        use      solve_cont_module_c
        use      iterdata_module_c
        use      hydro_module_c
        use      timedata_module_c
        use      bcs_dep_module_d
        use      scatter_module_d
        use      gather_module_d

C
*****

        implicit      none

#include "/home/cluster3/drockken/petsc-2.1.6/include/finclude/petsc.h"
#include      "/home/cluster3/drockken/petsc-
2.1.6/include/finclude/petscvec.h"
#include      "/home/cluster3/drockken/petsc-
2.1.6/include/finclude/petscmat.h"
#include      "/home/cluster3/drockken/petsc-
2.1.6/include/finclude/petscsles.h"
#include      "/home/cluster3/drockken/petsc-
2.1.6/include/finclude/petscvviewer.h"
#include      "/home/cluster3/drockken/petsc-
2.1.6/include/finclude/petscksp.h"

C#####
#

C      Other global and local variables.

C_____
-

C      Global variables being passed through the argument list.

        logical      (kind      =log_kind      ),
&      intent      (out      )
&      :: convin

        integer      (kind      =int_kind      ),
&      intent      (in      )
&      :: iterin_mn      ,iterin_mx,nnodemaxtot

        integer      (kind      =int_kind      ),
&      intent      (out      )
&      :: iconvout      ,iterin

```

```

integer (kind      =int_kind  )
&      :: iostatus

real (kind      =real_kind  ),
&      intent      (inout      )
&      :: test1

character (len      =*        ),
&      intent      (in          )
&      :: equation

real (kind      =real_kind  ),
&      dimension    (  nnodemax),
&      intent      (in          )
&      :: DPMIN      ,DPMAX
CHPFD_DISTRIBUTE DPMIN      (      _BLK_)
CHPFD_DISTRIBUTE DPMAX      (      _BLK_)

real (kind      =real_kind  ),
&      dimension    (  nnodemax),
&      intent      (inout      )
&      :: TEST2
CHPFD_DISTRIBUTE TEST2      (      _BLK_)

real (kind      =real_kind  ),
&      dimension    (nnodemaxtot)
&      :: NTMP_TOT
CHPFD_DISTRIBUTE NTMP_TOT   (      _BLK_)

real (kind      =real_kind  ),
&      dimension    (  nnodemax)
&      :: NTMP
CHPFD_DISTRIBUTE NTMP      (      _BLK_)

```

C

—

C Local variables.

```

logical (kind      =log_kind  )
&      :: convin_test1,convin_test2, acheck

integer (kind      =int_kind  )
&      :: inorth      ,ii,jj,its,its_mx,iksp_reason,
&      i,j,k1,k2,IERR, inz,onz,dnz,nn,nloc,k

integer (kind      = int_kind  )
&      ::  ATMP(1)      ! DTR

real (kind      =real_kind  )
&      :: resmax, rpgs2

real (kind      =real_kind  ), ! DTR
&      dimension    ( nconns )
&      :: DDPDXM,DDPDYM,DDPDZM
CHPFD_DISTRIBUTE DDPDXM      (      _BLK_)
CHPFD_DISTRIBUTE DDPDYM      (      _BLK_)

```

```

CHPFD_DISTRIBUTE DDPDZM      (      _BLK_)

      real      (kind      =real_kind ),      ! CTZ
      &          dimension    (      nnodemax)
      &          :: SCRATCHN4,  DELRHO, RESCLOLD, RESCOLD
CHPFD_DISTRIBUTE SCRATCHN4    (      _BLK_)
CHPFD_DISTRIBUTE DELRHO      (      _BLK_)

C_____
—

c      KSPConvergedReason reason

      Mat PTC_AA ! , PTC_PC
      PetscOffset I_P
c      PetscReal rel_tol
      PetscScalar P_ARRAY(1)
      PetscScalar TMP
      PetscViewer Viewer
      KSP ksp
      SLES sles
      Vec PTC_BB
      Vec PTC_P
c      Vec indexM, indexN
C_____
—

C#####
#

C
*****

C      Initialize some data.
c      print *, 'spc convin coming in'
      iterin      =0
      convin_test1=.false.
      convin_test2=.false.
      convin      =.false.
      DELTAP      =zero
      NTMP_TOT    =zero
      NTMP        =zero
      nn=nnodemaxtot

C
*****

C      Check if outer iteration is converged
C      Do this only when the inner it count of the mom. equation is one.

      acheck=.false.
      if (iterout.gt.1) then

```

```

c      print *, 'no mom its was ', Itary_uvw(iterout)
      if (Itary_uvw(iterout).eq.1) then
c      print *, 'setting acheck to true'
         acheck=.true.
      end if ! Itary_uvw
      end if ! iterout

      if (acheck) then
         if(Global_ALL(ABS(RESCL).LE.test1)) then
c      print *, 'setting this to iterin for test1'
            iterin=1
         end if

         if(Global_ALL(ABS(RESCL).LE.TEST2)) then
c      print *, 'setting this to iterin for TEST2'
            iterin=1
         end if

      end if ! acheck

C
*****

C      If the outer iteration is converged flag it so then exit.
C      Otherwise, tighten convergence criteria and solve.

      if(iterin.eq.1)then
c      wprint *,'spc immediately satisfying convin'
         iconvout=1
         return
      else
         iconvout=0
         test1=0.1*test1
         TEST2=0.1*TEST2
      end if      ! iterin.eq.1

C      Iteratively solve for the solution by successively guessing the
C      new values and computing residuals corresponding the the new
C      values.

C      create the unknown and right hand side vectors

      call VecCreateMPI(PETSC_COMM_WORLD,PETSC_DECIDE,nn,PTC_P,IERR)
      call VecSetFromOptions(PTC_P, IERR)

      call VecDuplicate(PTC_P, PTC_BB, IERR)

      inz = 6
      dnz = 7
      onz = 6

      call MatCreateMPIAIJ(
         & PETSC_COMM_WORLD, ! communicator
         & PETSC_DECIDE, PETSC_DECIDE, ! local
proc. sizes
         & nn, nn, ! global
rows, columns

```

```

        &                                dnz,PETSC_NULL_INTEGER,      ! max # of
nonzeroes per diagonal submatrix (limited by
c      number of connections per node plus one for diag term - also
null value to not use array option
        &                                onz,PETSC_NULL_INTEGER,      ! max # of
offdiagonals and null array term
        &                                PTC_AA, IERR)                ! specify matrix
and fortran error return val

c      this line isnt required when using matcreatempiaij
c      call MatSetFromOptions(PTC_AA, IERR)

      call SLESCreate(PETSC_COMM_WORLD, sles, IERR)

C
*****

c      Load the vector values. In parallel simulations the nodes are
c      distributed but the vector will be originally be global in
context
c      so we need to extract the global position number for the proper \
c      location within the matrix.

c      call VecGetOwnershipRange(PTC_BB, vstart, vend, IERR)
C      set the right hand side vector

      BB=BB ! *RDRESCDP
      do I=1,nnodemax
        TMP = BB(I)
        nloc = NODENOG(I)
        call VecSetValue(PTC_BB, nloc-1, TMP, INSERT_VALUES,IERR)
      end do

c      finish construction. This step is needed with VecSetValue(s)

      call VecAssemblyBegin(PTC_BB,IERR)
      call VecAssemblyEnd(PTC_BB,IERR)

c      call PetscViewerASCIIOpen(PETSC_COMM_WORLD,"vec-ori.bb",
c      & Viewer,IERR)
c      call PetscViewerSetFormat(Viewer, PETSC_VIEWER_ASCII_DEFAULT
c      & , IERR)
c      call VecView(PTC_BB, viewer,IERR)
c      call PetscViewerDestroy(viewer,IERR)
C
*****

c      Load the matrix diagonal coefficients into the matrix. In
c      parallel simulations the nodes are distributed but the matrix
will
c      be global in context so we need to extract the global position
c      number for the proper location within the matrix.

      A0=A0 ! *RDRESCDP
      do I=1,nnodemax
        TMP=A0(I)

```

```

        nloc = NODENOG(I)
        call MatSetValue(PTC_AA,nloc-1,nloc-1,TMP,INSERT_VALUES,IERR)
    enddo

C
*****

c      Load off diagonal coefficients. Store them in scalar variables
c      and then insert them into proper locations.

        do J=1,nconns

            k1      = NODEST(1,J)
            k2      = NODEST(2,J)

            TMP = A0T(1,J)
            call MatSetValue(PTC_AA,k1-1,k2-1,TMP,INSERT_VALUES,IERR)

            if (IERR.ne.izero) then
                print *, 'error', IERR; stop
            end if

            TMP = A0T(2,J)
            call MatSetValue(PTC_AA,k2-1,k1-1,TMP,INSERT_VALUES,IERR)

            if (IERR.ne.izero) then
                print *, 'error', IERR; stop
            end if

        end do

c      stop
c      call MatSetOption(PTC_AA, MAT_NO_NEW_NONZERO_LOCATIONS,IERR)

C
*****

C      Finalize assembly of matrix. Any intermediate work may be
c      performed between these statements.
c
        call MatAssemblyBegin(PTC_AA, MAT_FINAL_ASSEMBLY, IERR)
        call MatAssemblyEnd(PTC_AA, MAT_FINAL_ASSEMBLY, IERR)

c      call PetscViewerASCIIOpen(PETSC_COMM_WORLD,"matrix.default",
c      & Viewer,IERR)
c      call PetscViewerSetFormat(Viewer, PETSC_VIEWER_ASCII_DEFAULT
c      & , IERR)
c      call MatView(PTC_AA, viewer,IERR)
c      call PetscViewerDestroy(viewer,IERR)
c      stop

C
*****

C      Set operators of the matrix, in particular set the preconditioner
c      matrix to be derived from the matrix PTC_AA, and indicate that
c      the two will have the same nonzero pattern. Then, set the SLES

```

```

c      context.

      call SLEGetKSP(sles,ksp,IERR)

      call KSPSetTolerances(ksp,PETSC_DEFAULT_DOUBLE_PRECISION
&
,PETSC_DEFAULT_DOUBLE_PRECISION,PETSC_DEFAULT_DOUBLE_PRECISION,
&      iterin_mx,IERR)

      call SLESetOperators(sles, PTC_AA, PTC_AA,
&      SAME_NONZERO_PATTERN, IERR)

      call SLESetFromOptions(sles, IERR)

C
*****

C      solve the system

      call SLESSolve(sles, PTC_BB, PTC_P, ITS, IERR)

      iterin=ITS
c      print *, 'spc petsc its', iterin
C
*****

c      Update the solution variables.  The first array entry is located
c      at (I_P+1) per PETSc manual.

      call VecGetArray(PTC_P, P_ARRAY, I_P, IERR)

      do i=1,nnodemax
        DELTAP(I)=P_ARRAY(I_P+I)
      end do

C
*****

C      Apply boundary conditions to changes in pressure.

      call bc_s('change', PSOURCE, DELTAP)

C
*****

c      Limit DELTAP

      DELTAP=MIN(MAX(DELTAP,DPMIN),DPMAX)

C
*****

c      Under Relax

      DELTAP = 1.0*DELTAP

      rpgs2=one/pgs**2

```

```

C
*****

C      Gather changes in pressure from nodes to connection terminuses.

      call gathert_sn(DELTAP,DELTAPT)

C
*****

      SCRATCHM1=( RHOOLDT(1,:)*SOUNDOLDT(1,:)*DELTAPT(2,:)
&                +RHOOLDT(2,:)*SOUNDOLDT(2,:)*DELTAPT(1,:)
&                )
&                /
&                ( RHOOLDT(1,:)*SOUNDOLDT(1,:)
&                +RHOOLDT(2,:)*SOUNDOLDT(2,:)
&                )
      call gradbcm_sn(DELTAP,SCRATCHN1,SCRATCHM1,
& DDPDX,DDPDY,DDPDZ)

C


---



C      Compute changes in velocity.

C


---



C      Compute raw changes in velocity.

      DELTAU=-dt*rpgs2*DDPDY/MAX(RHO,tinyum)
      DELTAV=-dt*rpgs2*DDPDZ/MAX(RHO,tinyum)
      DELTAW=-dt*rpgs2*DDPDZ/MAX(RHO,tinyum)

      DELTAU=SIGN(MIN(ABS(DELTAU),0.25_real_kind*ABS(U)),DELTAU)
      DELTAV=SIGN(MIN(ABS(DELTAV),0.25_real_kind*ABS(V)),DELTAV)
      DELTAW=SIGN(MIN(ABS(DELTAW),0.25_real_kind*ABS(W)),DELTAW)

C


---



C      Apply boundary conditions to velocity changes.

      call bc_uvw('change',USOURCE,VSOURCE,WSOURCE,
& UWALL,VWALL,WWALL,
& DELTAU,DELTAV,DELTAW)

C


---



C      Gather velocity changes to connection terminuses.

      call gathert_vn(DELTAV,DELTAV,DELTAW,
& DELTAUT,DELTAVT,DELTAWT)

C
*****

```



```

C      Compute delta(q~) at median-mesh boundary.

C      CTZ Approach
C      -----

C      Compute changes in velocities at median mesh points (tilda velocs)

      call gradt_sm(DELTAPT,DDPDXM,DDPDYM,DDPDZM)

      DELTAUTM=-CNUM*DDPDXM*rpgs2
      DELTAVTM=-CNVM*DDPDYM*rpgs2
      DELTAWTM=-CNWM*DDPDZM*rpgs2

      if(1.eq.2)then

          DELTAT=zero
          DELTARHO=zero
          DELTARHOT=zero

      else ! 1.eq.1
C

```

```

C      Compute raw temperature changes.

      DELTAT=BBIGX*DELTAU+BBIGY*DELTAV+BBIGZ*DELTAW+CBIG*DELTAP

      DELTAT=SIGN(MIN(ABS(DELTAT),0.25_real_kind*ABS(T)),DELTAT)
      DELTAT=MERGE(MAX(DELTAT,zero),DELTAT,T.LE.zero)

C      Apply boundary conditions to temperature changes.

      if(flagtsource) then
          call bc_s('change',TSOURCE,DELTAT)
      end if ! flagtsource

C

```

```

C      Compute changes in density corresponding to changes in pressure
C      and temperature for the pressure-based scheme. Density changes
C      are given for the density-based scheme.

      DELTARHO=DRHODPT*DELTAP+DRHODTP*DELTAT

      DELTARHO=MIN(MAX(DELTARHO,-0.9_real_kind*RHO)
&                  ,10.0_real_kind*RHO)

C      Gather changes in density from nodes to connection terminuses.

      call gathert_sn(DELTARHO,DELTARHOT)

C      Now compute changes in density at connection median-mesh
C      boundary.

      DELTARHOM = zero

```

```

        DELTARHOM =half*(DELTARHOT(1,:)*(one+VFXUPW)
&                + DELTARHOT(2,:)*(one-VFXUPW))

        DELTARHOM=MAX(DELTARHOM,-half*RHOM)

    End if ! 1.eq.1
C

```

```

C
*****

C    Compute residual.

    RESCT(1,:)= VOLFLOWM*DELTARHOM
&            +RHOM*( DELTAUTM*AXM+DELTAVTM*AYM+DELTAWTM*AZM)
    RESCT(2,:)= -RESCT(1,:)

    call scattert_sn('add',RESC,RESCT)

    RESC= RESCL+DELTARHO+dt*( RESC
&            -RHOOUTFLOW*( DELTAU*AXN_OPEN
&                        +DELTAV*AYN_OPEN
&                        +DELTAW*AZN_OPEN
&            )-QFLOWOUT*DELTARHO)/MAX(VOL,tinyum)

C
*****

C    Apply boundary conditions to the residual.

    #if PRESSURE_BASED
        if(flagpsource) then
            call bc_s('change',PSOURCE,RESC)
        endif ! flagpsource
    #endif /* PRESSURE_BASED */
C        *** No need to call BC because we applied the boundary
C        *** condition to the derivative of the residual.

C
*****

C    Zero-out the residual at non-real nodes.

    RESC=MERGE(zero,RESC,NODETYPE.LT.0)

C
*****

    call VecRestoreArray(PTC_P, P_ARRAY, I_P, IERR)

C    stop
C

```

```

c          end if

C

```

```

C          Test for convergence.

          if(Global_ALL(ABS(RESC).LE.test1)) then
              convin_test1=.true.
c          print *, 'spc Convin1'
          endif ! global_all_resc

          if(Global_ALL(ABS(RESC).LE.TEST2)) then
              convin_test2=.true.
c          print *, 'spc convin2'
          endif ! global_all_RESC

C

```

```

          call VecDestroy(PTC_P, IERR)
          call VecDestroy(PTC_BB, IERR)
          call MatDestroy(PTC_AA, IERR)
          call SLESDestroy(sles, IERR)

C
*****

C          Set the outgoing convergence flags/variables and print necessary
C          messages.

          if(convin_test1.or.convin_test2) then
c          print *, 'spc convin_test1', convin_test1
c          print *, 'spc convin_test2', convin_test2
          convin=.true.
c          print *, 'spc convin = ', convin
          endif ! convin_test1_or_test2

          if(verbosity.ge.1) then
              if(convin_test1.and..not.convin_test2) then
                  Inout_String=' '
                  write(Inout_String,
& "(/, 'Matrix looks non-pos.-definite (equation=',a,')' /,a)")
& trim(equation),blank_line
                  call write_string(Inout_String,tty=.true.)
                  endif ! convin_test2_not_convin2
              endif ! verbosity

C
*****

c          print *, 'exiting solve_petsc_cont'
c          print *, '^^^^^^^^^^^^^^^^^^^^'
c          print *, '*****'
end

```

SOLVE_PETSC_CONT_PAR_2

Solves the native continuity equation using PETSc. This is the routine that was use for the majority of the calculations in this dissertation.

```
*DK solve_petsc_cont_par_2
      subroutine solve_petsc_cont_par_2(equation,iconvout,
      & iterin,iterin_mn,iterin_mx,
      & convin,test1,DPMIN,DPMAX,TEST2,nnodemaxtot)

C#####
C#

C      Purpose:

C          Solves a given scalar equation using PETSc

C      Input Variables:

C          equation      = The name of equation being solved.
C          iterin_mn      = Minimum number of inner iterations allowed.
C          iterin_mx      = Maximum number of inner iterations allowed.
C          test1          = A scalar number for checking for convergence.
C                        If the change in quantity being iterated on is
C                        less than or equal to this value during two
C                        successive iterations, the solution is
C      considered
C                        to be converged.
C          DPMIN          = Minimum allowable value of the scalar.
C          DPMAX          = Maximum allowable value of the scalar.
C          TEST2          = An array for checking for convergence. When
C                        every residual in every element is less than or
C                        equal to TEST2, the solution is considered to be
C                        converged.

C      Output Variables:

C          iconvout       = An integer flag for nonlinear (outer-iteration)
C                        convergence. A positive value of iconvout
C                        implies full nonlinear convergence, a zero
C                        value implies full linear convergence, and a
C                        negative value implies an impossible situation
C                        of achieving even linearized convergence at the
C                        current time step.
C          iterin         = Number of inner iterations required to converge.
C          convin         = A logical flag indicating if the solution
C                        got fully converged within the required number
C                        of iterations.

C#####
C#

C      Module list and other preliminaries.
```

```

#include "macros.HH"

        use      Kinds_module_c
        use      Parallel_module_c
        use      constants_module_c
        use      dimensions_module_c
        use      arrays_glo_module_c
        use      hydro_module_c
        use      options_module_c          , only:
&      flowtype
        use      physdata_module_c
        use      physics_module_c
        use      grads_module_d
        use      inout_module_d          , only:
&      blank_line  , Inout_String, write_string, open_file,
&      close_file
        use      inoutdata_module_c      , only:
&      verbosity
        use      solve_cont_module_c
        use      iterdata_module_c
        use      hydro_module_c
        use      timedata_module_c
        use      bcs_dep_module_d
        use      scatter_module_d
        use      gather_module_d
        use      anl_data_module_c      , only:
&      imatrixbug, Petsc_rtol, float_rtol, urf_p, use_urf

C
*****

        implicit      none

#include "/home/cluster3/drockken/petsc-2.1.6/include/finclude/petsc.h"
#include      "/home/cluster3/drockken/petsc-
2.1.6/include/finclude/petscvec.h"
#include      "/home/cluster3/drockken/petsc-
2.1.6/include/finclude/petscmat.h"
#include      "/home/cluster3/drockken/petsc-
2.1.6/include/finclude/petscsles.h"
#include      "/home/cluster3/drockken/petsc-
2.1.6/include/finclude/petscviewer.h"
#include      "/home/cluster3/drockken/petsc-
2.1.6/include/finclude/petscksp.h"

C#####
#

C      Other global and local variables.

C_____
-

C      Global variables being passed through the argument list.

        logical      (kind      =log_kind      ),
&      intent      (out      )
&      :: convin

```

```

integer (kind      =int_kind ),
&      intent      (in      )
&      :: iterin_mn ,iterin_mx,nnodemaxtot

```

```

integer (kind      =int_kind ),
&      intent      (out      )
&      :: iconvout  ,iterin

```

```

integer (kind      =int_kind )
&      :: iostatus, isize

```

```

real (kind      =real_kind ),
&      intent      (inout     )
&      :: test1

```

```

character (len      =*      ),
&      intent      (in      )
&      :: equation

```

```

real (kind      =real_kind ),
&      dimension    (  nnodemax),
&      intent      (in      )
&      :: DPMIN     ,DPMAX
CHPFD_DISTRIBUTE DPMIN      (      _BLK_)
CHPFD_DISTRIBUTE DPMAX      (      _BLK_)

```

```

real (kind      =real_kind ),
&      dimension    (  nnodemax),
&      intent      (inout     )
&      :: TEST2
CHPFD_DISTRIBUTE TEST2      (      _BLK_)

```

```

real (kind      =real_kind ),
&      dimension    (nnodemaxtot)
&      :: NTMP_TOT
CHPFD_DISTRIBUTE NTMP_TOT   (      _BLK_)

```

```

real (kind      =real_kind ),
&      dimension    (  nnodemax)
&      :: NTMP
CHPFD_DISTRIBUTE NTMP      (      _BLK_)

```

C

C Local variables.

```

logical (kind      =log_kind )
&      :: convin_test1,convin_test2, acheck

```

```

integer (kind      =int_kind )
&      :: inorth     ,ii,jj,its,its_mx,iksp_reason,
&      i,j,k1,k2,IERR, inz,onz,dnz,nn,nloc,k

```

```

integer (kind      = int_kind )
&      :: ATMP(1)    ! DTR

```

```

      real      (kind      =real_kind )
&      :: resmax, rpgs2

      real      (kind      =real_kind ), ! DTR
&      dimension ( nconns )
&      :: DDPDXM,DDPDYM,DDPDZM
CHPFD_DISTRIBUTE DDPDXM      (      _BLK_)
CHPFD_DISTRIBUTE DDPDYM      (      _BLK_)
CHPFD_DISTRIBUTE DDPDZM      (      _BLK_)

      real      (kind      =real_kind ), ! CTZ
&      dimension (      nnodemax)
&      :: SCRATCHN4, DELRHO, RESCLOLD, RESCOLD
CHPFD_DISTRIBUTE SCRATCHN4 (      _BLK_)
CHPFD_DISTRIBUTE DELRHO    (      _BLK_)

```

C_

```

c      KSPConvergedReason reason

```

```

      Mat PTC_AA ! , PTC_PC
      PetscOffset I_P
c      PetscReal rel_tol
      PetscScalar P_ARRAY(1)
      PetscScalar TMP
      PetscViewer Viewer
      KSP ksp
      SLES sles
      Vec PTC_BB
      Vec PTC_P
c      Vec indexM, indexN

```

C_

```

C#####
#

```

```

C
*****

```

```

C      Initialize some data.
c      print *, 'spc convin coming in'
      iterin      =0
      convin_test1=.false.
      convin_test2=.false.
      convin      =.false.
      DELTAP      =zero
      NTMP_TOT    =zero
      NTMP        =zero
      nn=nnodemaxtot

```

```

C
*****

C      Check if outer iteration is converged

      acheck=.false.

      if (iterout.gt.1) then
      if (Itary_uvw(iterout).eq.1) then
        acheck=.true.
      end if ! Itary_uvw
      end if ! iterout

      if (acheck) then
        if(Global_ALL(ABS(RESCL).LE.test1)) then
          iterin=1
        end if

        if(Global_ALL(ABS(RESCL).LE.TEST2)) then
          iterin=1
        end if

      end if ! acheck

C
*****

C      If the outer iteration is converged flag it so then exit.
C      Otherwise, tighten convergence criteria and solve.

      if(iterin.eq.1)then
        iconvout=1
        return
      else
        iconvout=0
        test1=0.1*test1
        TEST2=0.1*TEST2
      end if      ! iterin.eq.1

C      Iteratively solve for the solution by successively guessing the
C      new values and computing residuals corresponding the the new
C      values.

C      create the unknown and right hand side vectors

      call VecCreateMPI(PETSC_COMM_WORLD,PETSC_DECIDE,nn,PTC_P,IERR)
      call VecSetFromOptions(PTC_P, IERR)

      call VecDuplicate(PTC_P, PTC_BB, IERR)

      inz = 6
      dnz = 7
      onz = 6

      call MatCreateMPIAIJ(
&                                PETSC_COMM_WORLD,      ! communicator

```



```

        &                                PETSC_DECIDE, PETSC_DECIDE,      !      local
proc. sizes
        &                                nn, nn,                        !      global
rows, columns
        &                                dnz,PETSC_NULL_INTEGER,      !  max  #  of
nonzeroes per diagonal submatrix (limited by
c      number of connections per node plus one for diag term - also
null value to not use array option
        &                                onz,PETSC_NULL_INTEGER,      !  max  #  of
offdiagonals and null array term
        &                                PTC_AA, IERR)                  !  specify matrix
and fortran error return val

c      call MatCreate(PETSC_COMM_WORLD, PETSC_DECIDE, PETSC_DECIDE,
c      &              nn, nn, PTC_AA, IERR)

c      call MatSetFromOptions(PTC_AA, IERR)

      call SLESCreate(PETSC_COMM_WORLD, sles, IERR)

C
*****

c      Load the vector values.  In parallel simulations the nodes are
c      distributed but the vector will be originally be global in
context
c      so we need to extract the global position number for the proper \
c      location within the matrix.

c      call VecGetOwnershipRange(PTC_BB, vstart, vend, IERR)
C      set the right hand side vector

      do I=1,nnodemax
        TMP  = BB(I)
        nloc = NODENOG(I)
        call VecSetValue(PTC_BB, nloc-1, TMP, INSERT_VALUES,IERR)
      end do

c      finish construction. This step is needed with VecSetValue(s)

      call VecAssemblyBegin(PTC_BB,IERR)
      call VecAssemblyEnd(PTC_BB,IERR)

      if (imatrixbug) then
        call PetscViewerASCIIOpen(PETSC_COMM_WORLD,"vec-ori.dat",
&  Viewer,IERR)
        call PetscViewerSetFormat(Viewer, PETSC_VIEWER_ASCII_DEFAULT
&  , IERR)
        call VecView(PTC_BB, viewer,IERR)
        call PetscViewerDestroy(viewer,IERR)

      end if

C
*****

```

```

c      Load the matrix diagonal coefficients into the matrix.  In
c      parallel simulations the nodes are distributed but the matrix
will
c      be global in context so we need to extract the global position
c      number for the proper location within the matrix.

      do I=1,nnodemax
        TMP=A0(I)
        nloc = NODENOG(I)
        call MatSetValue(PTC_AA,nloc-1,nloc-1,TMP,INSERT_VALUES,IERR)
      enddo

C
*****

c      Load off diagonal coefficients. Store them in scalar variables
c      and then insert them into proper locations.

      do J=1,nconns
        k1 = NODEST(1,J)
        k2 = NODEST(2,J)
        TMP = A0T(1,J)
        call MatSetValue(PTC_AA,k1-1,k2-1,TMP,INSERT_VALUES,IERR)

        if (IERR.ne.0) then
          print *, 'matset-error 1'
          stop
        end if

        TMP = A0T(2,J)
        call MatSetValue(PTC_AA,k2-1,k1-1,TMP,INSERT_VALUES,IERR)

        if (IERR.ne.0) then
          print *, 'matset-error 2'
          stop
        end if

c      check
c      print *, PEInfo%thisPE,j,k1,k2,A0T(1,j),A0T(2,J)
      end do

c      stop
c      call MatSetOption(PTC_AA, MAT_NO_NEW_NONZERO_LOCATIONS, IERR)

C
*****

C
C      Add outlet velocities
      do i=1,nnodemax
        if(NODETYPE(i).eq.51)then
          TMP=(dt*AXN(i))/(RHO(i)*VOL(i)*4.)
          TMP=-TMP*dt/VOL(i)*AXN(i)*RHO(i)
          nloc = NODENOG(I)
          call MatSetValue(PTC_AA,nloc-1,nloc-1,TMP,ADD_VALUES,IERR)
          call MatSetValue(PTC_AA,nloc-2,nloc-1,-TMP,ADD_VALUES,IERR)

```

```

        endif
    enddo

C
*****
C
*****

C      Finalize assembly of matrix.  Any intermediate work may be
c      performed between these statements.

      call MatAssemblyBegin(PTC_AA, MAT_FINAL_ASSEMBLY, IERR)
      call MatAssemblyEnd(PTC_AA, MAT_FINAL_ASSEMBLY, IERR)

      if (imatixbug) then
      call PetscViewerASCIIOpen(PETSC_COMM_WORLD,"matrix.default",
&      Viewer,IERR)
      call PetscViewerSetFormat(Viewer, PETSC_VIEWER_ASCII_DEFAULT
&      , IERR)
      call MatView(PTC_AA, viewer,IERR)
      call PetscViewerDestroy(viewer,IERR)

      end if

C
*****

C      Set operators of the matrix, in particular set the preconditioner
c      matrix to be derived from the matrix PTC_AA, and indicate that
c      the two will have the same nonzero pattern.  Then, set the SLES
c      context.

      call SLESGetKSP(sles,ksp,IERR)

      if(float_rtol)then
      call KSPSetTolerances(ksp,petsc_rtol
&
,PETSC_DEFAULT_DOUBLE_PRECISION,PETSC_DEFAULT_DOUBLE_PRECISION,
&      iterin_mx,IERR)
      else
      call KSPSetTolerances(ksp,PETSC_DEFAULT_DOUBLE_PRECISION
&
,PETSC_DEFAULT_DOUBLE_PRECISION,PETSC_DEFAULT_DOUBLE_PRECISION,
&      iterin_mx,IERR)
      end if

      call SLESSetOperators(sles, PTC_AA, PTC_AA,
&      SAME_NONZERO_PATTERN, IERR)

      call SLESSetFromOptions(sles, IERR)

C
*****

C      solve the system

      call SLESSolve(sles, PTC_BB, PTC_P, ITS, IERR)

```

```

        iterin=ITS
c      print *, 'spc petsc its', iterin
C
*****

c      Update the solution variables.  The first array entry is located
c      at (I_P+1) per PETSc manual.

        call VecGetArray(PTC_P, P_ARRAY, I_P, IERR)

        do i=1,nnodemax
            DELTAP(I)=P_ARRAY(I_P+I)
        end do

C
*****

C      Apply boundary conditions to changes in pressure.

        call bc_s('change', PSOURCE, DELTAP)

C
*****

c      Limit DELTAP

        if (use_urfp) then
            DELTAP=DELTAP*urf_p
        end if
        DELTAP=MIN(MAX(DELTAP, DPMIN), DPMAX)

C
*****

        if (imatixbug) then
            call PetscViewerASCIIOpen(PETSC_COMM_WORLD,"vec-dp.dat",
&      Viewer,IERR)
            call PetscViewerSetFormat(Viewer, PETSC_VIEWER_ASCII_DEFAULT
&      , IERR)
            call VecView(PTC_P, viewer,IERR)
            call PetscViewerDestroy(viewer,IERR)

        end if

c      compute dependent terms and linear residual

        call res_lin_cont(.true.)

        if(imatrixbug.eq.1)then
            Inout_String=' '
            write(Inout_String,"(/,' Non-linear Norm :',e13.6
&      , ' Linear Norm      :',e13.6)
&      ")
&      sqrt(global_sum(abs(RESCL)**2))
&      ,sqrt(global_sum(abs(RESCL)**2))

```

```

        call write_string(Inout_String,tty=.true.)
    end if

c      O_RESCL=RESCL
c      O_RESC =RESC
c      call User_pst
    if (imatixbug) then

c      open(unit=473,file='res-mtx.dat')
c      do ii=1,nnodemax
c          write(473,*)ii,'nl,lin',RESCL(ii),RESC(ii)
c      end do
c      close(473)
c      stop
    end if

C
*****

c      call VecGetOwnershipRange(PTC_P,ii,jj,IERR)

c      k1=1
c      do I=ii,jj-1
c          NTMP(k1)=P_ARRAY(I_P+k1)
c          k1=k1+1
c      end do

c      call parallel_collate(NTMP_TOT,NTMP)
c      call parallel_dist(NTMP,NTMP_TOT)

c      DELTAP=NTMP+DELTAP

c      do I=1,nnodemax
c          print *, 'dp,nng', DP(I), NODENOG(I), PEInfo%thisPE
c      end do

c      call VecRestoreArray(PTC_P, P_ARRAY, I_P, IERR)

c      stop
C
-----

c      end if

C
-----

C      Test for convergence.

c      do ii=1,nnodemax
c          if(RESC(ii).gt.test1)then
c              print *, 'resc-test',ii,x(ii),y(ii),RESC(ii),test1
c          end if
c      enddo
c      if(Global_ALL(ABS(RESC).LE.test1)) then
c          convin_test1=.true.
c      endif ! global_all_resc

```

```

        if(Global_ALL(ABS(RESC).LE.TEST2)) then
            convin_test2=.true.
        endif ! global_all_RESC

C


---


        call VecDestroy(PTC_P, IERR)
        call VecDestroy(PTC_BB, IERR)
        call MatDestroy(PTC_AA, IERR)
        call SLESDestroy(sles, IERR)

C
*****

C      Set the outgoing convergence flags/variables and print necessary
C      messages.

        if(convin_test1.or.convin_test2) then
C          print *, 'spc convin_test1', convin_test1
C          print *, 'spc convin_test2', convin_test2
            convin=.true.
C          print *, 'spc convin = ', convin
        endif ! convin_test1_or_test2

        if(verbosity.ge.1) then
            if(convin_test1.and..not.convin_test2) then
                Inout_String=' '
                write(Inout_String,
& "(/,'Matrix looks non-pos.-definite (equation=' ,a,')' /,a)")
& trim(equation),blank_line
                call write_string(Inout_String, tty=.true.)
            endif ! convin_test2_not_convin2
        endif ! verbosity

C
*****

        end

```

USER_BCS

This routine is provided to illustrate how boundary conditions are set in CHAD.

```

8c8,9
<      & MFRACINFLOW,MFRACOUTFLOW)
---
>      & MFRACINFLOW,MFRACOUTFLOW,
>      & P_OPEN)
136a138
>
138a141
>      use          constants_module_c
139a143
>      use          nodetypes_module_c
142a147,149

```

```

>      use      anl_data_module_c      ,only:
>      &      ictz
>      use      Parallel_module_c      ! dtr
168a176
>      &      ,P_OPEN      ! CTZ
187a196
> CHPFD_DISTRIBUTE P_OPEN      (      _BLK_)
195a205,209
>      real      (kind      =real_kind )
>      &      ::dtrmu, dtra, dtry, dtrp,tmp1,tmp2,tmp3
>      &      ,dtmpl, dtmp2, dtmp3, dtmp4
>      integer    (kind      =int_kind )
>      &      ::ii, iid
219a234,299
>
***** C
> cdtr
> cdtr  We will use this section to apply a fully developed flow
profile
> cdtr  at the inlet for flow between two parallel plates.
>
> cdtr  Variables are defined as
> cdtr
> cdtr  dtrmu = anu0/rho = absolute viscosity
> cdtr  dtra  = distance between two plates
> cdtr  dtrp  = dp/dx
> cdtr  dtry  = y - position in the domain
>
> c      if(1.eq.2) then
>
> c      dtrp  = -0.6
> c      dtrmu = 0.05
> c      dtra  = 1.0
>
> c      tmp3  = dtrp/(2.*dtrmu)
>
> c      do ii=1,21
>
> CDTR  Set the node for which we wish to integrate the area about
> c      iid=1+81*(ii-1)
>
> CDTR  Set the lower y-coordinate of integration
> CDTR  If its the first node, set it to the coordinate for itself
> CDTR  Otherwise its the average of itself and the previous node.
>
> c      if(ii.eq.1)then
> c      dtmpl=0.0
> c      else
> c      dtmpl=( Y(iid)+Y(iid-81)) / 2
> c      end if
>
> CDTR  Set the upper y-coordinate of integration
> CDTR  If its the lat node, set it to the coordinate for itself
> CDTR  Otherwise its the average of itself and the next node.
>
> c      if(ii.eq.21)then
> c      dtmp2=1.0

```

```

> c      else
> c      dtmp2=( Y(iid+81)+Y(iid)) / 2
> c      end if
>
> CDTR Integrate the flow profile over the two node points
>
> c      VELINFLOW(iid)=
> c      &          tmp3*(dtmp2**3/3-dtmp2**2/2)
> c      &          -tmp3*(dtmp1**3/3-dtmp1**2/2)
>
> CDTR Then divide by cell height to get the average inflow velocity
>
> c      VELINFLOW(iid)= VELINFLOW(iid) / (dtmp2-dtmp1)
>
> CDTR Print it out to check it
> c      print *, 'high-low-diff', dtmp2, dtmp1, (dtmp2-dtmp1)
> c      print *, iid, x(iid),y(iid),VELINFLOW(iid)
> c      print *, ' '
> c c      end do
>
> c      stop
>
>
> c      end if
>
259a340,353
>
> C      P_OPEN is the pressure used to close the pressure integrals at
> C      boundaries          !      CTZ
>
>      if(ictz.eq.1)then
>
>          P_OPEN = hugenum+hugenum !      CTZ/DTR the addition same as
physics.ff
>
>          WHERE(NODETYPE.eq.ntp_outflow)
!      CTZ
>          P_OPEN=8.6115583684657e+4_real_kind
> cdtr      P_OPEN=pout
>          ENDWHERE
>      end if
>
262a357,358
>
>

```

USER_MONITOR

Calculates a global node number based upon cartesian coordinates.

```

*DK User_pst
      subroutine User_monitor

```

```

C#####
#

```



```

C      Purpose:

C      Calculates a global node number based upon cartesian coordinates.
C      Input Variables:

C          nnodemax      = Maximum value of a node number on this PE.
C          ICOLORN       = Node color after domain decomposition. That
C                        is, the processor number where the node will
C                        reside.
C          ICOLORE       = Element color after domain decomposition. That
C                        is, the processor number where the element will
C                        reside.
C          NODETYPE      = Node type (positive value implies a real node).

C      Output Variables:

C          imon_glo_no = Global node number corresponding to x-y-z
C          coordinates

C#####
C#

C      Module list and other preliminaries.

#include "macros.HH"

      use      Kinds_module_c
      use      Parallel_module_c      ,only:
&      global_maxval      ,
&      global_minval      ,
&      parallel_collate,
&      global_minloc
      use      arrays_glo_module_c
      use      constants_module_c
      use      dimensions_module_c
      use      inoutdata_module_c
      use      physdata_module_c
      use      hydro_module_c
      use      solve_hydro_module_c
      use      timedata_module_c
      use      anl_data_module_c      ! DTR
      implicit none

C#####
C#

C      Other global and local variables.

C_____
C_

C      Global variables being passed through the argument list.

C_____
C_

```

```

C      Local variables.

      real    (kind      =real_kind  ),
&      dimension (nnodemax)
&      :: RTMP

      integer (kind      =int_kind   )
&      :: ii

      integer (kind      =int_kind   ),
&      dimension (1)
&      :: RTMPX

C_____
-

C#####
#

C
*****

c      Compute the distance from each node to the requested monitoring
c      point location.

c      if (mon_x.gt.-99999) then
c          RTMP=SQRT((X-mon_x)**2+(Y-mon_y)**2+(Z-mon_z)**2)

c      find the global loc of the min RTMP
c          rtmpx=Global_Minloc(RTMP, MASK=NODETYPE.gt.0)

C
.....

C
_____

C
*****

      end

```

USER_PST

Writes user requested values to the post processing file.

```

170a171
>
175c176,177

```

```

<      &          parallel_collate
---
>      &          global_minval      ,
>      &          parallel_collate, scalar_sum, parallel_bcast
183a186,187
>      use          hydro_module_c
>      use          solve_hydro_module_c
187a192
>      use          anl_data_module_c      ! DTR
204a210,212
>      real      (kind      =real_kind      )
>      &          :: rtmpu, rtmpv, rtmpw, rtmpp,rtmpt,rtmpk,rtmpe      ! DTR
>
206c214
<      &          :: il      ,imat      ,iostatus      ,isp
---
>      &          :: il      ,imat ,iostatus      ,isp, ii, nnodemaxtot
214,218d221
<      real      (kind      =real_kind      ),
<      &          dimension      (      nnodemax)
<      &          :: SCRATCHN1
<      CHPDFD_DISTRIBUTE SCRATCHN1      (      _BLK_)
<
244,246c247,257
<      Inout_String=' '
<      Inout_String(1)='MACHNO      1'
<      call write_string(Inout_String,ttty=.false.,lun=nf_post)
---
> c      Inout_String=' '
> c      Inout_String(1)='RESUL      '
> c      call write_string(Inout_String,ttty=.false.,lun=nf_post)
> c      call write_array(lun=nf_post,fmt=gmv_format,
> c      &      limitexp=.true.,ARRAY=RESUL)
>
> c      Inout_String=' '
> c      Inout_String(1)='RESVL      '
> c      call write_string(Inout_String,ttty=.false.,lun=nf_post)
> c      call write_array(lun=nf_post,fmt=gmv_format,
> c      &      limitexp=.true.,ARRAY=RESVL)
248,250c259,269
<      SCRATCHN1=SQRT((U**2+V**2+W**2)/MAX(CSQ,tiny))
<      call write_array(lun=nf_post,fmt=gmv_format,
<      &      limitexp=.true.,ARRAY=SCRATCHN1)
---
> c      Inout_String=' '
> c      Inout_String(1)='O_RESC      '
> c      call write_string(Inout_String,ttty=.false.,lun=nf_post)
> c      call write_array(lun=nf_post,fmt=gmv_format,
> c      &      limitexp=.true.,ARRAY=O_RESC)
>
> c      Inout_String=' '
> c      Inout_String(1)='O_RESCL      '
> c      call write_string(Inout_String,ttty=.false.,lun=nf_post)
> c      call write_array(lun=nf_post,fmt=gmv_format,
> c      &      limitexp=.true.,ARRAY=O_RESCL)
262c281
<      outuser=.false.

```

```

---
> outuser=.true.
273c292,295
< filnamuser =postfile(1:il-1)//'_usr.' //filnam(1:3)
---
>
> c filnamuser =postfile(1:il-1)//'_usr.' //filnam(1:3)
>
> filnamuser ='monitors.dat'
307c329,334
< if(outuser) then
---
> if(1.gt.2)then
> rtmpu=Global_Maxval(U,MASK=NODENOG.eq.imon_node_u)
> rtmpv=Global_Maxval(V,MASK=NODENOG.eq.imon_node_v)
> rtmpw=Global_Maxval(W,MASK=NODENOG.eq.imon_node_w)
> rtmpt=Global_Maxval(T,MASK=NODENOG.eq.imon_node_t)
> rtmpp=Global_Maxval(P,MASK=NODENOG.eq.imon_node_p)
309,310c336,345
<
open_file(lun=nf_post,file=filnamuser,iostat=iostat,
< & status='unknown',form='formatted')
---
> c if(outuser) then
> c nnodemaxtot=scalar_sum(nnodemax)
> c if(imon_node_u.gt.0)then
> c do ii=1,nnodemax
> c if(imon_node_u.eq.NODENOG(ii))then
> c rtmpu=U(ii)
> c call parallel_bcast(rtmpu)
> c end if
> c end do
> c end if
312,317c347,354
< call write_array(lun=nf_post,fmt=gmw_format,
< & limitexp=.true.,ARRAY=X)
< call write_array(lun=nf_post,fmt=gmw_format,
< & limitexp=.true.,ARRAY=Y)
< call write_array(lun=nf_post,fmt=gmw_format,
< & limitexp=.true.,ARRAY=Z)
---
> c if(imon_node_v.gt.0)then
> c do ii=1,nnodemax
> c if(imon_node_v.eq.NODENOG(ii))then
> c rtmpv=V(ii)
> c call parallel_bcast(rtmpv)
> c end if
> c end do
> c end if
319c356,363
< call close_file(nf_post)
---
> c if(imon_node_w.gt.0)then
> c do ii=1,nnodemax
> c if(imon_node_w.eq.NODENOG(ii))then
> c rtmpw=W(ii)
> c call parallel_bcast(rtmpw)

```

```

> c          end if
> c          end do
> c          end if
321,322c365,372
<
open_file(lun=nf_post,file=filnamuser1,iostat=iostat,
<      &      status='unknown',form='formatted')
---
> c          if(imon_node_t.gt.0)then
> c      do ii=1,nnodemax
> c          if(imon_node_t.eq.NODENOG(ii))then
> c              rtmt=T(ii)
> c              call parallel_bcast(rtmt)
> c          end if
> c      end do
> c          end if
324,325c374,403
<          call write_array(lun=nf_post,fmt=gmw_format,
<      &      limitexp=.true.,ARRAY=CSQ)
---
> c          if(imon_node_p.gt.0)then
> c      do ii=1,nnodemax
> c          if(imon_node_p.eq.NODENOG(ii))then
> c              rtmpp=P(ii)
> c              call parallel_bcast(rtmpp)
> c          end if
> c      end do
> c          end if
>
>
>
open_file(lun=nf_post,file=filnamuser,iostat=iostat,
>      &      status='old',position='APPEND',form='formatted')
>
>      Inout_String=' '
>      write(Inout_String,"(i7,1x,
>      &          e13.6,1x,
>      &          e13.6,1x,
>      &          e13.6,1x,
>      &          e13.6,1x,
>      &          e13.6,1x,
>      &          e13.6)")
>      &      ncyc,
>      &      rtmpu,
>      &      rtmpv,
>      &      rtmpw,
> c      &      rtmt,
>      &      rtmpp,
> c      &      rtmpk,
> c      &      rtmpe,
>      &      time
>      call write_string(Inout_String,tty=.false.,lun=nf_post)
329d406
<      endif
330a408,409
> c      endif
>      end if ! 1.gt.2

```

USER_SRCS

Generates user defined source terms (such as buoyancy.)

```
117a118
>
125a127,131
>      use      Parallel_module_c
>      use      anl_data_module_c      , only:
>      &         ictz
>      use      gather_module_d ! DTR
>      use      scatter_module_d ! DTR
142c148
<      &         :: SCRATCHN1
---
>      &         :: SCRATCHN1, SCRATCHN2
143a150,156
> CHPFD_DISTRIBUTE SCRATCHN2      (      _BLK_)
>
>      real      (kind      =real_kind ),
>      &         dimension   (      nconns)
>      &         :: SCRATCHM1, SCRATCHM2
> CHPFD_DISTRIBUTE SCRATCHM1      (      _BLK_)
> CHPFD_DISTRIBUTE SCRATCHM2      (      _BLK_)
144a158,179
>      real      (kind      =real_kind ),
>      &         dimension   (      2,nconns)
>      &         :: SCRATCHT1, SCRATCHT2
> CHPFD_DISTRIBUTE SCRATCHT1      (_STR_, _BLK_)
> CHPFD_DISTRIBUTE SCRATCHT2      (_STR_, _BLK_)
>
>      real      (kind      =real_kind ),
>      &         dimension   (      : ),
>      &         allocatable
>      &         :: ARRAYX, ARRAYY, ARRAYZ
>
>      integer    (kind      =int_kind ),
>      &         dimension   (      : ),
>      &         allocatable
>      &         :: ARRAYT,ARRAYN
>
>      real (kind =real_kind )
>      &         tmp3, dtrp, dtrmu, dtra, tmp, darcy, forch, vmag,
>      &         vrfact, irfact, vmu,r,theta
>
>      integer    (kind      =int_kind )
>      &         ii,i,j,k, nid
227a263,276
>      tmp=0.0
>      do ii=1,nnodemax
>          if(x(ii).ge..15)      then
>          if(z(ii).ge.-0.116596) then
>          if(z(ii).le. 0.001596) then
>              tmp =tmp-25.*V01(ii)
```

```

>          SRCMOMZ(ii)=-25.*VOL(ii)
>      end if
>      end if
>      end if
>      enddo
> c      SPAREN(2,:)=NODETYPE
> c      SPAREN(3,:)=tmp
>
230a280
>

```

WRITE_MAX

Used for debugging by outputting the maximum variable values at a cell.

```

*DK write_max
      subroutine write_max(eqn)

C#####
C#

C      Purpose:

C      To debug the code by spitting out max variable values each call
C      Input Variables:

C      Input Variables

c      eq          = Integer flag to specify calling point

C#####
C#

C      Module list and other preliminaries.

#include "macros.HH"

      use      Kinds_module_c
      use      Parallel_module_c          ,only:
&      global_maxval          ,
&      global_minval
      use      arrays_glo_module_c
      use      constants_module_c
      use      dimensions_module_c
      use      eosdata_module_c          ,only:
&      mat_stress
      use      hydro_module_c
      use      inoutdata_module_c
      use      iterdata_module_c
      use      nodetypes_module_c
      use      options_module_c          ,only:
&      steady          ,hydrottype      ,intrfc_opt
      use      physdata_module_c
      use      physics_module_c

```

```

#if SIMPLE
    use          solve_uvw_module_c
#else /* SIMPLE */
    use          solve_hydro_module_c
#endif /* SIMPLE */
    use          timedata_module_c
    use          timerdata_module_c
    use          bcs_dep_module_d      ,only:
&              bc_uvw
    use          grads_module_d
    use          intrfc_module_d      ,only:
&              intrfc_s              ,
&              intrfc_uvw
    use          scatter_module_d
    use          symmetrize_module_d  ,only:
&              symmetrize_s4        ,
&              symmetrize_v
    use          timers_module_d
    use          anl_data_module_c    ,only:
&              iup_uvw,ictz

    implicit     none

C#####
#

C      Other global and local variables.

C_____
-

C      Global variables being passed through the argument list.

        integer      (kind      =int_kind      )
&                :: ireg          ,iterin_mn      ,iterin_mx, ii,k1,eqn,k2

C_____
-

C#####
#

        open(unit=29,file='maxvals.dat',status='old')

        if(eqn.eq.1)then
            write(29,*)'writing for momentum equation'
        elseif(eqn.eq.2)then
            write(29,*)'writing for cont equation'

        elseif(eqn.eq.3)then
        elseif(eqn.eq.4)then
        elseif(eqn.eq.5)then
        elseif(eqn.eq.6)then
        else
        end if

```



```

write(29,*)'max U ', global_maxval(U)
write(29,*)'max V ', global_maxval(V)
write(29,*)'max UTILDEM ', global_maxval(UTILDEM)
write(29,*)'max VTILDEM ', global_maxval(VTILDEM)
write(29,*)'max VOLFLOWM ', global_maxval(VOLFLOWM)
write(29,*)'max UM ', global_maxval(UM)
write(29,*)'max VM ', global_maxval(VM)
write(29,*)'max RHO ', global_maxval(RHO)
write(29,*)'max RHOM ', global_maxval(RHOM)
write(29,*)'max P ', global_maxval(P,MASK=NODETYPE.GE.0)
close(29)

C
*****

C
*****

end

```

WRITE_RESIDUALS

Writes normalized residuals to a file for plotting.

```

*DK write_residuals
  subroutine write_residuals

C#####
#

C    Purpose:

C          Writes normalized residuals to a file for convergence
monitoring

C    Input Variables:

C          None

C    Output Variables:

C          None

C#####
#

C    Module list and other preliminaries.

#include "macros.HH"

      use      Kinds_module_c
      use      Parallel_module_c      , only:
&      global_maxval      ,
&      global_minval      ,
&      global_sum

```

```

        use      arrays_glo_module_c
        use      anl_data_module_c          ,only:
&      plot_res,
&      norm_fact_u, norm_fact_v, norm_fact_w,norm_fact_t,
&      norm_fact_p, r_red_fact_u, r_red_fact_v, r_red_fact_w,
&      r_red_fact_t, r_red_fact_p
        use      constants_module_c
        use      dimensions_module_c
        use      hydro_module_c
        use      iterdata_module_c
        use      inoutdata_module_c
        use      nodetypes_module_c
        use      options_module_c
        use      physdata_module_c
        use      physics_module_c
        use      timedata_module_c
        use      inout_module_d            ,only:
&      Inout_String,close_file  ,inquire_file,open_file  ,
&      write_array ,write_string
        use      units_module_c            ,only:
&      ctemp
        use      solve_hydro_module_c      ,only:      ! DTR
&      RESUL,RESVL,RESWL,RESTL,RESCL
        implicit none

C#####
#

C      Other global and local variables.

C_____
—

C      Local variables.

        integer      (kind      =int_kind      )
&      :: k1,k2,ii, iostatus !dtr

C_____
—

C#####
#

c      We compute the normalization factor for the first 10 iterations
c      based upon the maximum nodal value at that time.  After 10
c      iterations, the normalization faction is the maximum of the
c      normalization factors within the first ten iterations.

c      This approach works well for closed domain problems but it would
be
c      better to base it off inflow terms for problems with inlets.

```

```

      Inout_String=' '

      write(Inout_String,"(i7,1x,e13.6,1x,
&                      e13.6,1x,
&                      e13.6,1x,
&                      e13.6,1x,
&                      e13.6,1x,
&                      e13.6,1x,
&                      e13.6)")

      &      ncyrc,
      &      r_red_fact_u,
      &      r_red_fact_v,
      &      r_red_fact_w,
      &      r_red_fact_t,
      &      r_red_fact_p,
      &      time

      call open_file(lun=56,file='residuals.dat',iostat=iostat,
&      status='old',position='APPEND',form='formatted')

      call write_string(Inout_String,tty=.false.,lun=56)

      call close_file(lun=56)

C
*****

      end

```

ZERO_WALL_TILDES

Zeroes the tilde velocities between two wall nodes.

```

*DK write_max
      subroutine zero_wall_tildes(TMPU,TMPV,TMPW)

C#####
#

C      Purpose:

C      Zeroes out the changes in tilde velocities when two nodes
C      are both wall nodes.

C      Input Variables

CDTR   TMPU      = Represents the change to UTILDEM
CDTR   TMPV      = Represents the change to VTILDEM
CDTR   TMPW      = Represents the change to WTILDEM
C      Output Variables

C#####
#

```

```

C      Module list and other preliminaries.

#include "macros.HH"

      use      Kinds_module_c
      use      arrays_glo_module_c
      use      constants_module_c
      use      dimensions_module_c
      use      physics_module_c

      implicit none

C#####
#

C      Other global and local variables.

C_____
-

C      Global variables being passed through the argument list.

      real      (kind      =real_kind  ),
      &          dimension  ( nconns    )
      &          :: TMPU, TMPV, TMPW
CHPFD_DISTRIBUTE TMPU      (      _BLK_ )
CHPFD_DISTRIBUTE TMPV      (      _BLK_ )
CHPFD_DISTRIBUTE TMPW      (      _BLK_ )

      integer    (kind      =int_kind   )
      &          ::      ii

C_____
-

C#####
#

cdtr  We must zero out the tilde velocities along walls.

      TMPU = TMPU *WALL_BDRY
      TMPV = TMPV *WALL_BDRY
      TMPW = TMPW *WALL_BDRY
c      print *, 'zeroing wall tildes'
c      stop
C
*****

C
*****

      end

```

ZERO_WALLS

Zeroes the tilde velocities between two wall nodes (alternate method to previous routine.)

```
      subroutine zero_walls(izwflag)

C#####
#

C      Purpose:

C      Zeroes out tilde velocities when two nodes are both wall nodes.

C      Input Variables

CDTR   izwflag      = A logical flag.  True indicates the boundary
CDTR                      connections must be found. False indicates
CDTR                      that they must be applied.
C      Output Variables

C#####
#

C      Module list and other preliminaries.

#include "macros.HH"

      use      Kinds_module_c
      use      Parallel_module_c      ,only:
&      global_maxval      ,
&      global_minval
      use      arrays_glo_module_c
      use      constants_module_c
      use      dimensions_module_c
      use      hydro_module_c
c      use      inoutdata_module_c
      use      iterdata_module_c
      use      nodetypes_module_c
      use      physdata_module_c
      use      physics_module_c
#if SIMPLE
      use      solve_uvw_module_c
#else /* SIMPLE */
      use      solve_hydro_module_c
#endif /* SIMPLE */
c      use      timedata_module_c
c      use      timerdata_module_c

      implicit none

C#####
#

C      Other global and local variables.
```

```

C_____
-
C      Global variables being passed through the argument list.

      integer      (kind      =int_kind      )
      &            :: tmp, ii,k1,k2,izwflag

C_____
-
C#####
#

cdtr  For each connection we must determine if each node is a wall node

      if (izwflag) then

          WALL_BDRY=1.

          do ii=1,nconns

              k1=NODEST(1,ii)
              k2=NODEST(2,ii)

cdtr  If both nodes of a connection are free_slip nodes, and flow is
cdtr  Prohibited via CONTAB then flag the connection as a wall.

              tmp=CONTAB(1,k1)+CONTAB(4,k1)+CONTAB(6,k1)+
      &          CONTAB(1,k2)+CONTAB(4,k2)+CONTAB(6,k2)
              if (
c      &          (NODETYPE(k1).eq.ntp_freeslip) .and.
c      &          (NODETYPE(k2).eq.ntp_freeslip) .and.
      &          (tmp.eq.0.)
      &          ) then

                  WALL_BDRY(ii)=0.
              end if
          end do
      else

cdtr  We must zero out the volumetric flow rate along walls.
          print *, 'zeroing walls lets stop'
          stop
          UTILDEM = UTILDEM*WALL_BDRY
          VTILDEM = VTILDEM*WALL_BDRY
          WTILDEM = WTILDEM*WALL_BDRY
          VOLFLOWM= VOLFLOWM*WALL_BDRY

          end if

C
*****

```

```
C
*****
end
```

Author's Biography

Daniel T. Rock was born on October 3, 1973 in a suburb of Cleveland Ohio. In 1992 he enrolled at the University of Cincinnati's College of Engineering five year long Nuclear Engineering program. As part of a mandatory cooperative education program there, he spent six months doing probabilistic risk assessment software development at Science Applications International Corporation. Later, he spent time at the Fernald Environmental Management Project Mixed Waste group. He graduated in the summer of 1997.

Daniel then enrolled at the University of Illinois' Nuclear Engineering program. There, he began working with Dr. Rizwan Uddin doing computational analyses using VIPRE , STAR-CD and CFX. Together, they co-authored several papers discussing the need for advanced computational analyses in nuclear industry with Dr. David Weber and Dr. Constantine Tzanos at Argonne National Laboratory.

Having enjoyed his work during his master's degree, Daniel continued into doctoral research with computational fluid dynamics. Together, they worked on the CHAD software code, looking to improve the performance of the continuity solver, and overall user experience with the code. Aspects of their work has been presented at several ANS conferences and published in the associated proceedings.

Daniel currently works as a nuclear and thermal-hydraulic methods engineer for Global Nuclear Fuel – Americas in Wilmington, NC.