**SANDIA REPORT**

SAND96-2331 • UC-405
Unlimited Release
Printed September 1996

# MPSalsa
# A Finite Element Computer Program for Reacting Flow Problems
## Part 2 – User's Guide

A. Salinger, K. Devine, G. Hennigan, H. Moffat, S. Hutchinson, J. Shadid

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550
for the United States Department of Energy
under Contract DE-AC04-94AL85000

SAND96-2331
Unlimited Release
Printed September 1996

Distribution
Category UC-405

# MPSalsa

## A FINITE ELEMENT COMPUTER PROGRAM FOR REACTING FLOW PROBLEMS

## PART 2 - USER'S GUIDE[1,2]

A. Salinger[3], K. Devine[4], G. Hennigan[3], H. Moffat[5], S. Hutchinson[3], J. Shadid[3]

Sandia National Laboratories
Albuquerque, New Mexico 87185

**Abstract Follows**

1. This document can be downloaded from: http://www.cs.sandia.gov/CRF/mpsalsa.html
2. This work was partially funded by Department of Energy, Mathematical, Information, and Computational Sciences Division, and was carried out at Sandia National Laboratories, operated for the US Department of Energy under contract no. DE-AC04-94AL85000.
3. Parallel Computational Sciences Department (org. 9221).
4. Parallel Computing Sciences Department (org. 9226).
5. Chemical Processing Science Department (org. 1126).

## Acknowledgments

We would like to thank Professor Michael Jensen for preparing a number of the fluid mechanics examples and for urging the development of the output routines, and Aaron Thomas for benchmarking an early version of the code. We would also like to thank Ed Boucheron for identifying many instances of undesirable functionality so that we could remove them from the code. Finally, we would like to thank Rod Schmidt for his careful reading of this document.

## Abstract

This manual describes the use of MPSalsa, an unstructured finite element (FE) code for solving chemically reacting flow problems on massively parallel computers. MPSalsa has been written to enable the rigorous modeling of the complex geometry and physics found in engineering systems that exhibit coupled fluid flow, heat transfer, mass transfer, and detailed reactions. In addition, considerable effort has been made to ensure that the code makes efficient use of the computational resources of massively parallel (MP), distributed memory architectures in a way that is nearly transparent to the user. The result is the ability to simultaneously model both three-dimensional geometries and flow as well as detailed reaction chemistry in a timely manner on MP computers, an ability we believe to be unique.

MPSalsa has been designed to allow the experienced researcher considerable flexibility in modeling a system. Any combination of the momentum equations, energy balance, and an arbitrary number of species mass balances can be solved. The physical and transport properties can be specified as constants, as functions, or taken from the Chemkin library and associated database. Any of the standard set of boundary conditions and source terms can be adapted by writing user functions, for which templates and examples exist.

The user can choose between a steady-state solution, an accurate transient run, a pseudo-transient method for relaxing stiff steady-state problems, and a continuation run for analysis of the system's steady-state behavior with respect to a parameter.

Through the input file, the user has considerable control over the nonlinear and linear solution strategies in order to find the fastest and most robust method for solving a given problem. The nonlinear solver includes an inexact Newton method and a backtracking strategy. For solving linear systems, a number of Krylov-based iterative methods along with several choices for preconditioners are available through the Aztec library.

A large set of example problems is included in Appendices to familiarize the user with the capabilities and choices within MPSalsa. These examples serve to illustrate MPSalsa capabilities and to provide a variety of input files to use as templates for closely related application problems. Many of these examples can be run on a single processor or on multiple parallel processors.

**Table of Contents**

## 1. Introduction

In this report, the practical details and interface for running the suite of computer codes called MPSalsa are presented, along with a number of example problems. A companion theory manual provides the equations and solution methodology [42]. Employing unstructured meshes on massively parallel (MP) computers, MPSalsa is designed to solve two- or three-dimensional problems that exhibit coupled fluid flow, heat transport, species transport, and chemical reactions. The equations defined in MPSalsa for fluid flow and mass conservation are the momentum transport and the total mass continuity equations for incompressible or variable density Newtonian fluids (Navier-Stokes equations). The heat transport equation and an arbitrary number of species transport-reaction equations are coupled with each other through chemical reaction source terms and with the fluid flow equations through property variation and body force terms.

MPSalsa employs unstructured grids, using the ExodusII finite element database for its input and output files [40]. Therefore, it can be used in conjunction with the CUBIT mesh generation package [24], as well as other mesh generation packages that support the ExodusII standard. A number of pre- and post-processing routines for the ExodusII database can be used. Currently, two- and three-dimensional grids with Cartesian coordinates are supported.

From its inception, MPSalsa has been designed for distributed memory MIMD computers with thousands of processors. It also runs on traditional serial workstations and networks of serial workstations. Interprocessor data communication and global synchronization are accomplished by a small number of message passing routines. These routines have been ported to many different message passing protocols, including the MPI standard [34] and the native nCUBE and Intel Paragon protocols. To achieve efficient parallel execution, the unstructured finite element mesh is partitioned or load-balanced in a pre-processing step. Each processor is assigned nodes from the mesh such that the computational load is balanced and the total amount of information communicated between neighboring processors is minimized. A general, automated method for subdividing an unstructured computational mesh is necessary. An ad-hoc or by-hand method would prove to be unusable for large meshes, and the resulting parallel communication efficiency would be difficult to predict, assess and control. In our implementation, we have used a general graph and mesh partitioning utility, Chaco [22], developed at Sandia National Laboratories.

MPSalsa uses a finite element (FE) method to approximate the solution to the transport equations for momentum, total mass, thermal energy, and individual gas-phase chemical species. The approach is designed for low Mach number flows where an algorithm employing an implicit coupling between the pressure and velocity field is required. The discretization method is a Petrov-Galerkin finite element method (PGFEM) with pressure stabilization [25]. For more highly-convective flows that are still laminar, a streamwise-upwinding (SUPG) stabilization is available [3, 48]. Each processor is responsible for calculating updates for all the unknowns at each of its assigned FE nodes. Each processor also stores and performs operations on the rows in the fully-summed, distributed matrix associated with these unknowns. Along processor subdomain boundaries, replicated FE unknowns, called "ghost unknowns," are stored and updated through interprocessor communication. These ghost unknowns are quantities needed for the local residual calculation and matrix-vector multiplication on a processor. Interprocessor communication occurs for each step of the iterative solution of the linear system as well as for each outer step in the non-linear and time-transient algorithms. This communication constitutes the major unstructured interprocessor communication cost in the program, and its algorithm has been extensively optimized within MPSalsa [43].

MPSalsa includes the option of using the Chemkin library to provide rigorous treatment of ideal-gas multicomponent transport, including the effects of thermal diffusion [28]. Chemical reactions occurring in the gas phase and on surfaces are also treated by calls to Chemkin [28] and Surface Chemkin [5], respectively. Thus, MPSalsa can handle varying numbers and types of chemical reactions and species in a robust manner. For example, the code can handle the complex temperature and pressure dependence predicted for unimolecular reactions (using the Troe parameterization [14]), important for chemical vapor deposition (CVD) systems which typically run at sub-atmospheric pressures. Surface site fractions and bulk-phase mole fractions are defined on all reacting surfaces using the Surface Chemkin package. Through this method, complex Langmuir-Hinshelwood-type [30] and precursor adsorption surface mechanisms, characteristic of many real CVD and catalysis surface systems, can be incorporated into the reacting flow analysis code. The capability of modeling simple dilute species transport and reaction, without the need of linking to Chemkin, is also included in MPSalsa.

Both steady and transient flows may be analyzed. The time integration methods include true transient, pseudo-transient, and steady implicit solvers. The steady solver can be driven by a continuation routine for efficient parameter study of a system. A fully-implicit, fully-coupled Newton routine is implemented for robustness. The Jacobian matrix includes all coupling between the equations and unknowns, and neglects only terms due to the variation of physical properties calculated by Chemkin. A full numerical Jacobian that includes all terms is also available. The nonlinear solver has additional features for speed and robustness, including an inexact Newton approach and a backtracking algorithm.

After construction of the distributed sparse matrix, the FE application calls the Aztec library of parallel, preconditioned Krylov solvers [26, 43, 44]. On each processor, the solvers operate on the local distributed sparse matrix and local solution vector using a combination of global structured communication and unstructured communication to implement the parallel solver kernels. A substantial set of preconditioners is available, including several versions of ILU factorization, a domain-decomposition method. Although these advanced preconditioners require considerable memory, they provide a huge gain in robustness.

Solution output from the program is achieved through several means. Output can be written to either a standard serial ExodusII file format [40] or a parallel extension of the ExodusII file format [23]. This extension consists of an individual standard serial ExodusII file for each processor with extra arrays that map the local numbering scheme on an individual processor to the global numbering scheme and encode the necessary communication information. The format can be used on both MP computers, such as the Intel Paragon, and distributed computing systems, such as groups of workstations. This parallel I/O capability can be used with today's primitive parallel I/O facilities with nearly linear speedup. A small but growing number of specialized output functions that analyze the solution and write solution information in non-ExodusII formats have been written for specific applications.

This report serves to document the user interfaces within MPSalsa and to provide several example problems. Chapter 2 describes several important pre-processing steps needed to carry out numerical simulations in an MP environment and the user interfaces to them. Section 2.1 gives a general description of mesh generation capabilities for ExodusII meshes. Section 2.2 describes how to run "exoIIlb," an ExodusII interface to the Chaco package described above. Section 2.3 describes how to set up and run Chemkin. Section 2.4 describes the pre-processor, "guacamole," which is used to set up and manipulate the ExodusII serial output file. Section 2.5 describes the serial and parallel I/O capabilities of the code. Section 2.6 gives some information on how to compile the code, and Section 2.7 shows how to run it.

MPSalsa is controlled by a large input file, in which the user can change everything from the number of processors to the convergence criteria for the linear solver routine. Chapter 3 describes the MPSalsa input file line by line. For instance, the problem type, which indicates which equations are to be solved, is specified in the General Specifications section, described in Section 3.1. Material properties and equations of state are described in Section 3.6. MPSalsa has extensive facilities for incorporating boundary conditions, which are documented in Section 3.7.

The user can extend the models past what has been pre-defined within MPSalsa [42]. Functions can be written to represent variations in physical properties, additional source terms, and special boundary conditions, any of which can be dependent on the current solution, position, or time. In addition, functions can be written for specifying an initial guess, for testing the MPSalsa solution with an analytic solution, and for specifying a continuation parameter. The interfaces to these routines are described in Chapter 4.

Chapter 5 involves a general discussion of some solution strategies that can help the user tune MPSalsa for a specific application. MPSalsa implements a number of advanced numerical solution procedures for solving systems of nonlinear PDEs. The optimal choice of these methods can be difficult and, thus, we include a section to aid in this selection. Section 5.1 describes strategies for reaching steady-state solutions. There are many choices and parameters in the MPSalsa input file that control the solution algorithm and can greatly effect speed, convergence behavior, and robustness. This chapter is intended to introduce the user to some of these options.

Appendix A lists and describes some user functions for application-specific boundary conditions and output routines (e.g., Danckwerts' boundary condition and time history output).

The next three appendices contain example problems. Appendix B covers four simple examples with mass transfer, most of which can be run on a single processor. Appendix C covers a set of fluid mechanics and heat transfer problems on refined two-dimensional meshes. Appendix D contains three models for Chemical Vapor Deposition (CVD) reactors which involve flow, heat transfer, and mass transfer on three-dimensional meshes.

# 2. Pre-Processing and Running MPSalsa

This chapter details the steps needed to run a successful MPSalsa simulation. It is recommended that the user first try this process with some example problems before starting on a new problem. There are several preprocessing steps that need to be done for every new mesh before running the MPSalsa program itself. They reflect the added complexities of conducting numerical simulations in a massively parallel computing environment. These steps include mesh generation, load balancing (only for multi-processor problems), and running the "guacamole" pre-processor for setting up the serial ExodusII output file and checking the input file for errors. For problems that get information from the Chemkin library, the Chemkin interpreter must also be run to create input files for the Chemkin suite of subroutines.

## 2.1. Mesh Generation

MPSalsa uses the ExodusII [40] finite element database format for storing the mesh and solution information. The FASTQ [1] package can be used to generate two-dimensional meshes, and either CUBIT [24] or FASTQ with GEN3D [17] can be used to create three-dimensional meshes. All of the mesh generation is done on workstations.

During mesh generation, parts of the mesh are grouped as separate element blocks and identified with an integer element block ID. In the Materials Specifications section of the MPSalsa input file, the element block IDs of the computational domain are associated with a material, which may have different transport properties and constitutive models than other materials. Not all element blocks created in the mesh generation and stored in the ExodusII mesh file need be associated with a material, in which case such element blocks are not included in MPSalsa's computational domain. Note, however, that severe load imbalances may result, since load balancing is currently conducted only over all element blocks defined in an ExodusII file.

All surfaces where boundary conditions will be applied must be identified as node sets or side sets during mesh generation. The application of boundary conditions is simpler if all surfaces that share the same boundary conditions for all equations are grouped into the same node set or side set. A node set is a list of nodes, while a side set contains sides of elements. Node sets can have Dirichlet conditions applied to them, but cannot support Neumann or Mixed conditions which require integration over the surface. Side sets may have all types of boundary conditions applied (Dirichlet, Neumann, or Mixed), since the elemental information is available to compute surface integrals.

## 2.2. Mesh Partitioning / Load Balancing

When running MPSalsa on more than one processor, the mesh is partitioned into subdomains so that each processor "owns" a set of nodes. To assure that the work load is balanced among the processors, an equal number of nodes is assigned to each processor. At the same time, an optimal partition will minimize the amount of interprocessor communication needed to build the finite element residuals and Jacobian matrix by grouping neighboring nodes together on one processor.

The Chaco [22] package, developed at Sandia, is a general graph partitioning program. We use the application "exoIIIb" to run Chaco to partition the nodes of a finite element mesh stored in the ExodusII database. The "exoIIIb" program creates partitioning information and writes it in a load-balance file (with a ".nemI" extension) in the NemesisI format [23]. (Note that this interface to the load balancer is new as of May, 1996, so many of the example problems have load balance files with the old naming convention, including the ".exoII" extension.) The load balance file contains information about the nodes owned by each processor and about "ghost nodes," which are owned by another processor but needed for residual calculations. With this information, the communication pattern for updates of ghost nodes for the mesh may be generated without any interprocessor communication.

The utility "exoIIIb" is run on a serial workstation and requires either command line parameters or a small input file to specify the name of the ExodusII mesh, the number of processors to partition it into, and the partitioning method. There are a variety of options for the partitioning algorithm, but we generally use the multilevel method [21]. An example of the input file, often called "input-ldbl," is shown in Figure 2.1. The only lines that are commonly changed are the input ExodusII file name and the number of partitions (processors), which is expressed as a product of two integers on the last line. Although any pair of integers whose product is 32 would also partition the mesh for 32 processors, the 4x8 designation would minimize communication for running on a rectangular set of processors that has dimensions 4x8. For hypercube-based machines, the argument for the `Machine Description` line may be designated as HYPERCUBE = *n*, where *n* is the dimension of the hypercube.

Additional options for "exoIIIb" parameters, including how to visualize the resulting mesh decomposition, may found in the "exoIIIb" manual page, the Chaco User's Guide [22], and the Nemesis User's Guide [23].

To partition the mesh, type the following command:

```
> exoIIIb -a input-ldbl
```

```
INPUT EXODUSII FILE        = box200.exoII
GRAPH TYPE                 = NODAL

DECOMPOSITION METHOD       = MULTIKL, NUM_SECTS=1
SOLVER SPECIFICATIONS      = TOLERANCE=2.0e-3,USE_RQI,RQI_VMAX=200

MACHINE DESCRIPTION        = MESH= 4x8
```

*Figure 2.1. Sample input file, usually named "input-ldbl," for the* `exoIIlb` *load balancing command.*

The load-balance file created from the file in Figure 2.1 will be named "box200-m32-bKL.nemI." The root name is the same as the ExodusII mesh file, the "m" signifies a mesh architecture, followed by the number of processors, while the "bKL" term refers to the multilevel method [21] with Kernighan-Lin improvement [29]. For information on the partitioning algorithm, see the Chaco [22] and NemesisI [23] manuals.

## 2.3. Chemkin Interpreter

Kinetic and transport data, such as the mixture viscosity, mixture thermal conductivity, multicomponent diffusion coefficients, and reaction rates, can be computed using the Chemkin library [28]. If Chemkin is to be used, information on the species and reactions for both the gas and surface phases must be supplied in the Chemkin and Surface Chemkin [5] input files. We use the convention that these files have ".gas" and ".sur" extensions. For example, the mechanism for the deposition of silicon nitride from $SiF_4$ and $NH_3$ in $H_2$ carrier gas is contained in the files "Si3N4.gas" and "Si3N4.sur." These input files must be interpreted once to form linking files that can be efficiently read into MPSalsa. The current version that is installed in MPSalsa, ChemkinII, creates binary linking files, so the interpretor must be rerun on every new machine.

A utility shell script called "interp" for executing the interpreters on a front-end workstation or the MP machine itself has been created and resides in the "bin" directory for each machine and operating system (e.g., "$MPSALSA_HOME/arch/sgi/bin/interp" for an SGI workstation, and "$MPSALSA_HOME/arch/smos/bin/interp" for SUNMOS, where $MPSALSA_HOME is the directory in which all MPSalsa libraries and utilities have been installed). For all machines, interp can be run on the command line followed by the root name of the Chemkin data files, for instance:

```
> interp Si3N4
```

for the silicon nitride mechanism. On the Intel Paragon, it can be run this same way using the "paragon" executable (for the OSF operating system) or using the "smos" executable (for SUNMOS).

The "interp" command is a script that runs three separate interpreters: "ckinterp" for the gas-phase chemistry mechanism, "skinterp" for the surface-phase chemistry mechanism, and "tranfit" for the dilute multicomponent gas-phase transport properties [5, 27, 28]. Several recent publications include further details and examples of application programs using the Chemkin libraries [6, 7, 33].

The "interp" utility creates three linking files needed for MPSalsa execution: "chem.bin," "surf.bin," and "tran.bin." In addition, two links to databases are created ("tran.dat" and "therm.dat"). The other files that are created are not needed. The names of the three "*.bin" files can be changed, but they must be specified in the Chemistry Specifications section of the input file (see Section 3.4).

When "interp" is run on a workstation, copies of the "*.bin" linking files are also created: "chem.bin.ws," "surf.bin.ws," and "tran.bin.ws." The "guacamole" pre-processor, described in Section 2.4, automatically adds the ".ws" extension to the file names given in the input file before looking for the files. The Chemkin binary files created on a parallel machine will not overwrite the ".ws" files, so "guacamole" can be run on one processor using the same input file as the parallel run.

MPSalsa will soon be upgrading to the newest Chemkin version, ChemkinIII, which allows for the creation of ASCII (and, therefore, machine-independent) linking files, which will greatly simplify the use of the interpreter.

## 2.4. Guacamole

A pre-processing routine called "guacamole" runs on a single processor and has two main purposes: to error-check the input file and to produce a serial ExodusII output file, creating fields and header information for user-defined output variables. This utility uses the same I/O routines as MPSalsa. The command for executing the pre-processor is

```
> guacamole <input-file>
```

where <input-file> is the name of the MPSalsa input file, and is, by default, "input-salsa." The executable is normally in the "bin" directory for the current workstation, so for an SGI workstation, the executable is "$MPSALSA_HOME/arch/sgi/bin/guacamole." The preprocessor sets up header information in the ExodusII output file, which requires that all variable information be predefined. However, once the variables are defined, time series data of arbitrary size may be efficiently output to the ExodusII file.

When "guacamole" creates the ExodusII output file, it writes all the mesh information to the file and creates space for the output of the solution variables. Therefore, whenever the mesh changes or the number of variables changes, "guacamole" must be rerun. For example, if a user has been running a fluid-mechanics problem (Problem Type `fluid_flow`) and decides to add the energy equation (Problem Type `fluid_flow_energy`) and request output of the temperature unknowns, "guacamole" must be rerun. It must also be rerun if the user redefines the selection of solution components to be included in the output file.

If "guacamole" is not run to generate the output file and scalar output of the results is requested, then MPSalsa will quickly terminate with the message:

```
check_output_specs: WARNING, output file "output_file.exoII" does not exist!
[ex_open] Error: failed to open output_file.exoII read only
   exerrval = -1
ERROR returned from ex_open on Processor 0!
```

## 2.5. Serial and Parallel I/O Utilities

MPSalsa may be run using either serial (i.e., scalar) or parallel I/O facilities. The least complicated way to run MPSalsa is by using the scalar input – scalar output mode. A diagram of what is involved is included in Figure 2.2. As an initial step, "guacamole" is run to create the serial ExodusII output file. The pre-processor "guacamole" parses the MPSalsa input file to determine the user's choice of variables to output. When Chemkin is being used, "guacamole" also parses the Chemkin linking files to obtain the number of gas-phase species and their character string names.

The user is now ready to run MPSalsa in scalar I/O mode, either on one or on many processors. In MPSalsa, processor 0 first reads the MPSalsa input file and, when Chemkin is to be used, the Chemkin linking files. This information is broadcast to all processors. Then, processor 0 reads the serial load balance file, and its information is broadcast to all nodes and processed in parallel. Once this step is done, each processor knows which nodes it "owns," and additionally, which nodal information it needs from other processors. Processor 0 then reads the ExodusII mesh file and broadcasts its information to all processors. Each processor searches the messages for mesh information that it needs. Finally, each processor renumbers elements and nodes contiguously in its local memory. Local-to-global mapping vectors are retained for output processing.

Alternatively, MPSalsa can do I/O on the parallel file system using the NemesisI package [23], as depicted in Figure 2.3. The parallel format is a multiple file format, with the number of

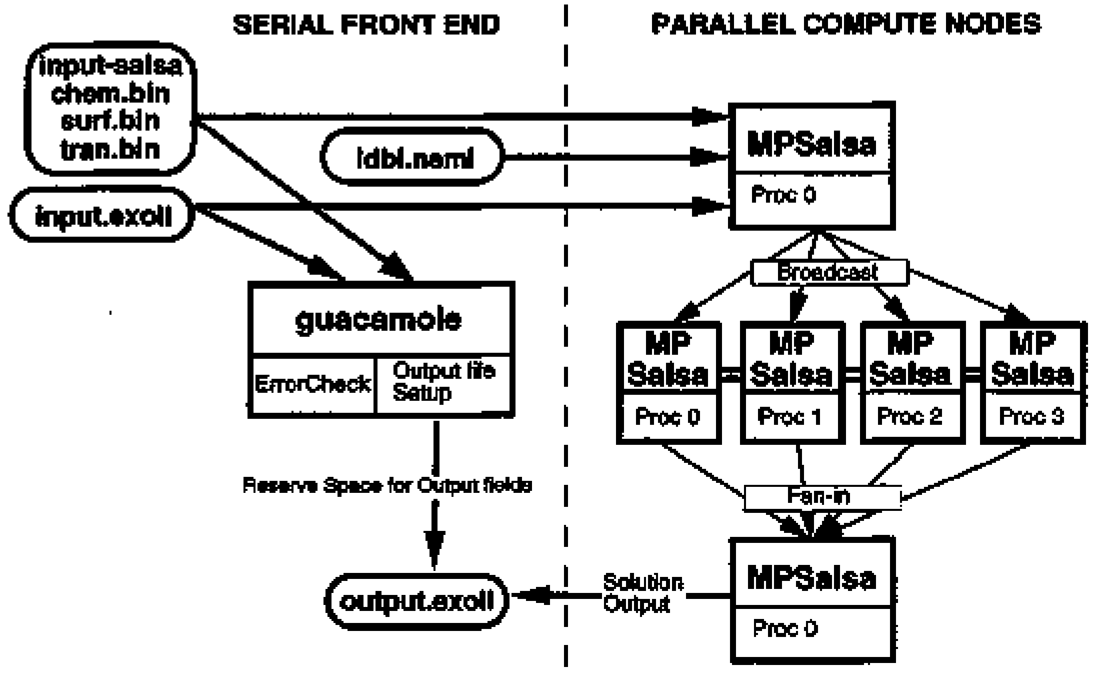


*Figure 2.2. Scalar Input - Scalar Output mode for I/O. The Broadcast and Fan-in routines have the potential to create I/O bottlenecks.*

files equaling the number of processors. A file name's suffix denotes which processor owns the file. The file structure within each parallel file is similar to the serial format, with the addition of local-to-global mapping information. It includes all load-balancing information contained in the serial load balance file as well as all information needed to set up the local computing environment on a processor, including ghost-node and communication information.

The parallel I/O capability is enabled in MPSalsa via compilation flag options. The pre-processor "guacamole" must be run to include user-defined header information in the output file. The "ex2pex" utility, part of the NemesisI package [23], is run next on the parallel computer. It translates the serial ExodusII file into the parallel file format and stores the parallel files on the file system to be used for MPSalsa's parallel I/O. It requires exactly the number of processors that will be used in the subsequent MPSalsa calculation. When MPSalsa is executed, processor 0 reads the input file and broadcasts its information to all processors as in the serial I/O case. However, in the parallel I/O case, each processor then reads its own parallel ExodusII file to initialize the mesh. Parallel solution output occurs in a reverse fashion, with each processor writing its own portion of the solution vector to its own output file.

For visualization of results, results in a set of parallel ExodusII files must be collected to a serial ExodusII file. A utility "pex2ex" is currently being developed that will automatically combine parallel ExodusII files into one ExodusII file. Until it is completed, however, two

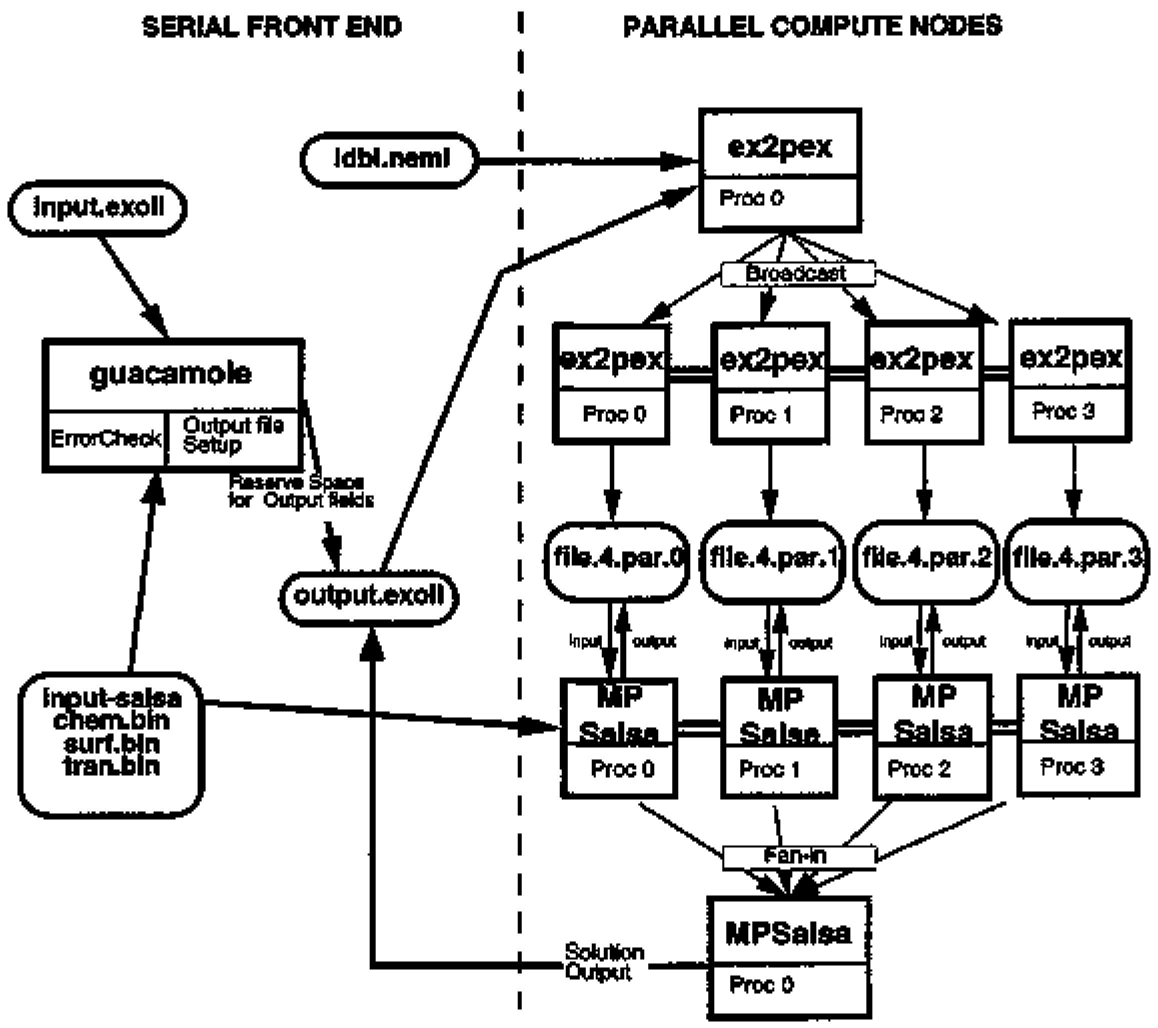


*Figure 2.3. Parallel I/O capabilities of MPSalsa.*

methods of obtaining serial ExodusII files exist. Both serial and parallel output may be specified for the run (see Section 3.9). This option will produce complete ExodusII files containing results from all time steps on both the serial and parallel file systems. If only the final result in a set of parallel ExodusII files is desired, the user can restart MPSalsa using the final result as the initial condition read from the parallel file system (see Section 3.8), maintaining the same stopping criteria as were used in the original computation, and specifying serial output. MPSalsa takes one Newton step to recognize that the stopping criteria are satisfied and writes the result to the serial file system.

## 2.6. Compiling MPSalsa

MPSalsa can be compiled on a number of different architectures. The GNU "make" program should be used to process the two-level Makefile structure. Machine-specific makefiles have been created since the message-passing routines, compiler names, and compiler options vary between machines. The source code usually is installed in a directory named "*/Salsa." This directory usually has the following files and subdirectories (identified by appending "/"):

```
> ls
  CVS/                 Obj_ncube/        Obj_sgi/        Obj_sun/
  CVS-CFile-Header     Obj_ncube_ps/     Obj_sgim4/      el/
  CVS-MFile-Header     Obj_paragon/      Obj_smos/       md/
  Makefile             Obj_paragon_ps/   Obj_smos_ps/    pe/
  Obj_alpha/           Obj_puma/         Obj_sol/        ps/
  Obj_hp/              Obj_puma_ps/      Obj_sp2/        rf/
```

The source code for MPSalsa is stored in the last five subdirectories, which have two-character names. The directories starting with "Obj_*" hold the compiled object files, dependency files, and the executable ("salsa") for a specific machine/operating system. All of the parallel machines have the additional option of compiling for parallel I/O, for which there are the separate directories with the "_ps" suffix.

To compile for a specific machine/operating system, the GNU make utility "gmake" is used. The target is the same as the extension on the "Obj_*" directory. For example, to compile for a Silicon Graphics workstation, a user would type

```
> gmake sgi
```

in the "*/Salsa" directory. To compile for the Intel Paragon with the SUNMOS operating system, a user would type

```
> gmake smos
```

on a workstation that has cross-compilers installed.

MPSalsa runs on top of several software packages. Before MPSalsa may be linked, these packages must be compiled and stored in architecture-dependent directories. For example, the following directories are used to store libraries, include files, and binaries for SGI computers: $MPSALSA_HOME/arch/sgi/lib, $MPSALSA_HOME/arch/sgi/include, $MPSALSA_HOME/arch/sgi/bin, where $MPSALSA_HOME is the directory in which all MPSalsa libraries and utilities have been installed. Pointers to these directories are included in the top level MPSalsa Makefile. The first I/O package needed is NetCDF [37], the underlying format of the ExodusII unstructured finite element package [40]. ExodusII is the next package that needs to be installed. The other I/O package needed is NemesisI [23], the parallel extension to ExodusII. In addition, the Chemkin libraries [5, 27, 28] are needed if the user wants to use this database for ideal gas transport and gas- and surface-phase reactions. The Chaco package is need for load balancing [22]. MP linear solvers within MPSalsa are implemented in the Aztec package [26], which in turn needs to have the Y12 package of sparse matrix linear solver routines [49]. Aztec, as well as a few of the other packages, require LAPACK [31] and BLAS [4] as well.

## 2.7. Running MPSalsa

The successful compilation of MPSalsa results in the creation of an executable in the machine-dependent subdirectory, "*/Salsa/Obj_xxx/salsa." MPSalsa can be run on workstations by executing the program with the input file name as the argument, i.e.,

```
> salsa <input-file>
```

The default input file name is "input-salsa."

On the Intel Paragon with the SUNMOS version of MPSalsa (whose executable is in the "Obj_smos" subdirectory), MPSalsa can be executed with the following command,

```
> yod -sz <np> salsa <input-file>
```

where *np* is the number of processors. The value of *np* must agree with the number of processors specified in the input file and the number of processors that the mesh was partitioned for. Execution of the "yod" command will spawn an MPsalsa job in the compute partition of the Paragon. As described in Section 2.5, either a serial file or a set of parallel files on the parallel file system must have been initialized previously for solution output to occur. Best results are obtained when both the executable and the I/O files are stored on local Paragon disks, rather than on nfs-mounted disks.

## 3. The Input File

In MPSalsa, problem-specific parameters are specified through an input file, which has the default name of "input-salsa." The input file is organized into 11 sections. The inclusion of certain sections is mandatory (General, Solution, Solver, Material, Boundary Condition and Initial Condition/Guess Specifications); other sections are optional (Enclosure Radiation, Output, Parallel I/O and Function Data Specifications). The Chemistry Specifications section is required only for problem types for which mass balance equations are solved (see Table 3.1). Each section is identified by MPSalsa by a unique section header, shown between two dashed lines in all of the examples below. MPSalsa does not parse a section unless it can find the section's header. If a required section's header is not found, MPSalsa generates an error message and exits. If an optional section's header is not found, no error message is generated.

Each section is made up of several lines. Each line consists of a keyword followed by an equals sign and arguments that can be strings, integers, flags, or real numbers. In this chapter, each line of input is described and the type of acceptable argument is given in italics. When there are a small number of choices for an argument, such as `yes` or `no`, they are represented using the format `{yes | no}`. Optional text is listed between square brackets, such as `[`*`int`*`]`, and input lines that are optional are completely enclosed in square brackets. For these input parameters, MPSalsa assigns the default value that is specified in the text.

### 3.1. General Specifications

General aspects of an MPSalsa execution are specified in the General Specifications section of the input file. Items such as the type of equations to be solved and the number of processors to be used in obtaining a solution are given in this section. This section is required and must begin with the General Specifications header, as illustrated in Figure 3.1.

```
----------------------------------------------------------------
General Specifications
----------------------------------------------------------------
Problem type                      = whole_enchilada
Input FEM file                    = cvd-reac1.exoII
LB file                           = cvd-reac1-m256-bKL.nemI
Output FEM file                   = cvd-reac1-out.exoII
Number of processors              = 256
Cartesian or Cylindrical when 2D = Cartesian
Stabilization                     = default
Debug                             = 3
```

*Figure 3.1. General Specifications example section.*

`Problem type =` *string*

The problem type input file line tells MPSalsa which equations are to be solved. MPSalsa can solve the Navier-Stokes equations in conjunction with the continuity equation, an energy equation, and an arbitrary number of species mass balance equations. Currently being tested are equations for flow in porous media and the $k$ and $\varepsilon$ equations for modeling turbulent flow, which will be detailed in future releases of this document. Equations for modeling plasma and electromagnetism may be incorporated in the future, as may the capability of using a pre-computed velocity field in the convective terms of the energy and species transport equations (for decoupled physics).

The current strings recognized by MPSalsa and the equations that they enable are listed in Table 3.1.

| Equation Type → | Momentum | Total Mass | Energy | Species Mass |
|---|---|---|---|---|
| Number of Equations in Type → | $N_{Dim}$ | 1 | 1 | $N_S$ |
| Problem Type ↓ | | | | |
| `fluid_flow` | X | X | | |
| `energy_diff` | | | X | |
| `mass_diff` | | | | X |
| `fluid_flow_energy` | X | X | X | |
| `fluid_flow_mass` | X | X | | X |
| `energy_mass_diff` | | | X | X |
| `whole_enchilada` | X | X | X | X |

*Table 3.1. The seven currently recognized strings for the* `Problem Type` *input file line are listed, and the governing equation types that each flag enables are indicated. The number of equations associated with each type is shown in the second row, where $N_{Dim}$ is the number of spatial dimensions in the problem and $N_S$ is the number of species, specified in Section 3.6.*

`Input FEM file =` *filename*

This line specifies the name of the input ExodusII file containing the FEM geometry information. This file usually has a ".exoII" extension. It can include a path specification. This file must exist prior to the run.

[LB file = *filename*]

This line specifies the name of the load-balance file for runs to be performed on more than one processor. It can include a path specification. The file must be in the NemesisI format, and usually has a ".nemI" extension. (Older files have the ".exoII" extension.) This input line is read only for runs performed on multiple processors. Default = none; error if not specified for multi-processor runs.

[Output FEM file = *filename*]

This line specifies the name of the ExodusII output file. This file is also used to provide initial solution data for restarts, which are specified on the `Set Initial Condition/Guess` input file line in Section 3.8. The file name can include a path specification. This ExodusII file must exist prior to the run, having been generated by the "guacamole" preprocessor (see Section 2.4). Visualization of the FE solution uses this file. This input line is used only for scalar I/O; for runs utilizing parallel I/O, special file names are generated (see Section 3.10). Default = none; error if not specified for restarts or runs with scalar output.

[Number of processors = *integer*]

This line is used to specify the number of processors that will be utilized in solving the problem. For multiprocessor runs, this number must match the number of processors that the mesh was partitioned for. Default = `1`.

[Cartesian or Cylindrical when 2D = *string*]

This line specifies what coordinate system to use for 2D problems. Currently the only valid value is `Cartesian`. Future choices will include `Cylindrical_2` and `Cylindrical_3` for axisymmetric problems with two or three momentum balances to be solved. Default = `Cartesian`.

[Interpolation order = *string*]

This line specifies the interpolation order for all quantities in the finite-element model. Valid options are `linear` and `quadratic`. Default = `linear`.

[Stabilization = {default | supg}]

There are currently two choices for stabilization of the FE equations: `default` and `supg`. The `default` option is a pressure-stabilized Petrov Galerkin method [25, 48], which allows the use of equal-order interpolation of the pressure and velocity primitive variables. The `supg` option activates the streamwise-upwinding Petrov-Galerkin stabilization scheme [3] in addition to the pressure stabilization. Streamwise upwinding improves convergence to highly-convected solutions (high Reynolds number flows) and reduces the amplitude of oscillations in the solution. Default = `default`.

`[Debug = ` *integer*`]`

This line specifies how much information should be output to *stdout* during the run of MPSalsa, as well as how much summary information the linear solver library should output. The value of *integer* must lie in the range [0, 10], with 2 being a common value. Examples are:

| | |
|---|---|
| `Debug = 0` | Minimal info is printed to *stdout*; only a summary of important flags and entries into important code segments are printed. |
| `Debug > 0` | Along with the above information, timing information and summary information on the global FE model (not the local processor FE model), node sets, side sets, and boundary conditions are printed. The solver library prints out residual summaries as well. |
| `Debug > 6` | Along with the above information, summary information on the local processor FE model is printed. Processor-based vector quantities such as residual, initial guesses, and solutions are included. Processor-based communication summaries and local-to-global mapping information are also printed. |
| `Debug > 9` | Along with the above information, information on the local matrix is printed. This can be a significant amount of information and is really meant to debug smaller problems in detail. |

Default = `2`.

## 3.2. Solution Specifications

The Solution Specifications section of the input file allows the user to choose the desired solution type, such as steady-state or time-dependent, and to control aspects of the solution procedure, such as the time step size. This section of the input file is mandatory and must begin with the Solution Specifications header, as shown in Figure 3.2.

```
-----------------------------------------------------------------
Solution Specifications
-----------------------------------------------------------------
Solution Type                        = transient
Order of integration/continuation    = 1
Step Control                         = on
Relative Time Integration Error      = 4.0e-3
Initial Parameter Value              = 100.0
Initial Step Size                    = 1.0e-5
Maximum Number of Steps              = 75
Maximum Time or Parameter Value      = 100.0
```

*Figure 3.2. Solution Specifications section example.*

In the rest of this section, each line of the Solution Specifications section is described separately. Since time-dependent and continuation runs both take steps from one solution to the next, many of the lines have dual meanings depending on the solution type.

`Solution Type =` *string*

This line specifies the type of solution desired, which can be one of the following five strings: `steady`, `transient`, `pseudo`, `continuation`, and `optimization`. If the `steady` string is specified, the code will attempt to solve the steady-state version of the governing equations (with no time derivative terms). The rest of this section of the input file is then ignored.

When the solution type is `transient` or `pseudo`, the time-dependent equations will be solved. A `transient` run attempts to follow the solution in a time-accurate manner by keeping the integration error under a specified tolerance, while the `pseudo` option is used to time step to a steady state (or past uninteresting transient behavior) by aggressively increasing the time step size regardless of the error in the time integrator. The specifics of the integration and stepping scheme can be manipulated with the subsequent input file lines.

The `continuation` solution type is used to solve for a series of steady-state solutions as a function of a parameter. The steady-state versions of the governing equations are solved, the continuation parameter is incremented and a new steady-state solution is sought. The subsequent lines in this section are used to control the run. The user has the flexibility of choosing any combination of physical properties and boundary condition values as the continuation parameter, but must do so by programming the routine `user_continuation` in file "rf_user_continuation.c" and recompiling (see Section 4.7 and Section 5.4).

The `optimization` solution type is not currently a supported feature, but has been used successfully for one application [8]. This solution type is similar to continuation, but instead of a single parameter being incremented within MPSalsa, a set of parameters is changed by an external optimization program. MPSalsa must be modified to calculate and write out an objective function after every solution for the optimization package to use.

`[Order of integration/continuation =` *integer*`]`

This flag has separate meanings depending on whether the solution type is time-dependent (`transient` or `pseudo`) or `continuation`. For `transient` or `pseudo` solutions, this flag has a value of `1` for first-order Forward-Euler/Backward-Euler predictor/corrector integration, and a value of 2 for a second-order Adams-Bashforth/Trapezoid-Rule scheme. (The second-order scheme starts with pair of first-order steps to get started.) Default = `1`.

For `continuation` runs, this flag can have a value of `0`, `1`, or `2`. A value of `0` turns on zero-order continuation, where the solution at the previous step is used as an initial guess for the current step. (This is equivalent to changing the value of the continuation parameter in the input file and restarting from the previous solution.) A value of `1` selects first-order (or Euler-Newton) continuation. In this case, the tangent to the previous solution with respect to the continuation parameter is calculated numerically, and is used to calculate an initial guess for the current solution. For problems whose solutions vary linearly with respect to the continuation parameter, this guess should be the correct solution. A flag value of 2 selects arc-length continuation, which is not currently implemented. This option will allow the user to follow steady-state solution branches that pass through turning points with respect to the continuation parameter. Default = `1`.

`[Step Control = {on | off}]`

The `Step Control` input line is read for `transient`, `pseudo`, and `continuation` solution types, and can have values of `on` or `off`. When step control is on, the step size will be adjusted after successful steps. For `transient` runs, the step size is chosen as a function of the value of the `Relative Time Integration Error` (described below). For `pseudo` and `continuation` runs, the step size will always be increased following a successful step, with the increase depending on the ratio of the number of Newton iterations needed for convergence divided by the maximum number of Newton iterations allowed. If the value of the `Step Control` is `off`, the step size is never increased. For any of the solution types and either of the flag values, the step size is cut in half after a failed step (i.e., when a converged solution is not found in the maximum number of Newton iterations). Default = `on`.

[Relative Time Integration Error = *float*]

The Relative Time Integration Error input line is used only for transient solutions. This line sets the target for the error incurred on each time step. A value of the time integration error is calculated using the difference between the predicted and corrected value of the solution by the method of [20]. If this estimated error is twice the value set in the input file, the time step is rejected and the time step size is cut in half. Otherwise, if Step Control is on, the ratio of the input error value and the estimated error are used to pick the next step size. The value of the Relative Time Integration Error must be greater than the Solution Relative Error Tolerance, which is input in the Solver Specifications section to set the convergence criterion for the linear solver. Default = $10^{-3}$.

[Initial Parameter Value = *float*]

The Initial Parameter Value input line is used only for continuation runs. The number is the initial value of the continuation parameter. See Section 4.7 for details on the implementation of continuation. Default = none, which is an error for continuation runs.

[Initial Step Size = *float*]

The Initial Step Size input line is used for transient, pseudo, and continuation runs. The value is the size of the first time step for time integration runs and the first continuation parameter step size for continuation runs. When Step Control is off, this step size stays constant throughout the run as long as each step converges. Default = none, which is an error for transient, pseudo, and continuation runs.

[Maximum Number of Steps = *integer*]

This input line is used for transient, pseudo, and continuation runs. When this maximum number of steps is reached, the program will terminate. Default = 1000.

[Maximum Time or Parameter Value = *float*]

This input file line is used for transient, pseudo, and continuation runs. When this value is exceeded by the time value in time-dependent runs or the continuation parameter in continuation runs, the program will terminate. Default = none.

## 3.3. Solver Specifications

The Solver Specifications section of the input file controls the nonlinear and linear solver routines used in MPSalsa. It is a required section of the input file. An example of this section, including the Solver Specifications header, is found in Figure 3.3. Each line is discussed below.

```
-------------------------------------------------------------------
 Solver Specifications
-------------------------------------------------------------------
Override Default Linearity Choice              = default

-- nonlinear solver subsection:

Number of Newton Iterations                    = 15
Use Modified Newton Iteration                  = no
Enable backtracking for residual reduction     = no
Choice for Inexact Newton Forcing Term         = 4
Calculate the Jacobian Numerically             = no
Solution Relative Error Tolerance              = 1.0e-3
Solution Absolute Error Tolerance              = 1.0e-8

-- linear solver subsection:

Solution Algorithm                             = gmres
Convergence Norm                               = 0
Preconditioner                                 = no_overlap_ilu
Polynomial                                     = LS,1
Scaling                                        = row_sum
Orthogonalization                              = classical
Size of Krylov subspace                        = 25
Maximum Linear Solve Iterations                = 50
Linear Solver Normalized Residual Tolerance    = 1.0e-6
```

*Figure 3.3. Solver Specifications section example.*

[`Override Default Linearity Choice =` *string*]

This input line can be set to three possible strings: `default`, `linear`, or `nonlinear`. The code decides whether the set of governing equations are linear or nonlinear depending on the problem type specified at the top of the input file. For instance, an `energy_diff` problem is assumed to be linear, while a `fluid_flow_energy` problem is assumed to be nonlinear. If users decide to override this default, as would be needed, for example, when using a temperature-dependent thermal conductivity with an otherwise linear heat equation, they can set the flag to `linear` or `nonlinear`. Default = `default`.

### 3.3.1. Nonlinear Solver Subsection

`[Number of Newton Iterations =` *integer*`]`

This line specifies the maximum number of Newton iterations that MPSalsa will allow in a single nonlinear solve. If this maximum is reached and the convergence criterion has not been met, the nonlinear solve ends unsuccessfully. For steady-state problems, MPSalsa terminates with a fatal error. For time-dependent problems, a convergence error is triggered for the current time step, and control is returned to the time stepping routine. Currently, the time stepping routine reverts to a Backward Euler method, halves the time step, and tries again. Similarly for continuation problems, the continuation algorithm cuts the parameter step-size in half and attempts to resolve the problem. Default = 25.

`[Use Modified Newton Iteration = {yes | no}]`

A modified Newton iteration uses a previously-computed preconditioning matrix for the Newton step, instead of recomputing the preconditioner from the Jacobian at the current solution. This option is not yet supported. Default = no.

`[Enable backtracking for residual reduction = {yes | no | default}]`

When a Newton iteration causes the norm of the residual to increase rather than decrease, backtracking will not accept the update. Instead, the algorithm looks in the same direction as the solution update from the Newton iteration. Performing residual calculations along the solution path given by this direction, it finds the solution that minimizes the residual [9, 10]. Backtracking has been shown in some cases to help converge to a steady-state when Newton's method without backtracking failed. The `default` flag disables backtracking for `transient` runs but enables backtracking for all other solution types (`pseudo`, `steady`, and `continuation`). Default = `default`.

`[Choice for Inexact Newton Forcing Term =` *integer*`]`

An inexact Newton's method uses Newton's method with an iterative linear solver, where the linear solver method (e.g., GMRES) is not forced to fully converge at each step. The reasoning behind this method is that it is a waste of computational time to fully solve the linear system when the nonlinear system itself is far from a converged solution. Inexact Newton steps are controlled by a single parameter, *eta_k*, which is the required drop in the ratio of the norm of the residual to the initial norm of the residual for a given linear solve. A normal Newton's method uses a small, constant value for *eta_k* so that each linear solve is accurate, as it would be when using a direct solver. This is the case when the inexact Newton forcing term is set to 4, with the *eta_k* tolerance value given by the `Linear Solver Normalized Residual Tolerance` input line below. Other values for the inexact Newton forcing term, 0–3, allow for larger values of *eta_k*, so that each Newton iteration takes less time; however, more Newton iterations are often required for convergence. The possible values for the flag are summarized in Table 3.2. Default = `0`.

| Flag Value | Choice for *eta_k* in Inexact Newton's Method |
|---|---|
| `0-1` | Eisenstat and Walker, Method 1 [9, 10] |
| `2` | Eisenstat and Walker, Method 2a |
| `3` | Eisenstat and Walker, Method 2b |
| `4` | Linear Solver Normalized Residual Tolerance ("Exact Newton") |

*Table 3.2. This table summarizes the choices for the* Inexact Newton forcing term. *The variable eta_k is the required drop in the linear residual for a successful linear solve.*

```
[Calculate the Jacobian Numerically = {yes | no}]
```

A fully numerical Jacobian may be used in MPSalsa for debugging purposes. Instead of the Jacobian matrix being computed analytically, the residual equations for each element are recomputed one extra time for each unknown in the element while that unknown is numerically perturbed. A forward difference formula is used to calculate the Jacobian contributions. For problems with large numbers of unknowns per node, the numerical Jacobian can be more than an order of magnitude slower than the analytic Jacobian, in part because rigorous property evaluations for multicomponent gas equations are very expensive. The numerical Jacobian is a powerful tool for debugging changes to the governing equations as well as for checking the effect of physical property variations -- some of which are ignored in the analytic Jacobian but included in the numerical one -- on the convergence behavior. Default = `no`.

```
[Solution Relative Error Tolerance = float]
[Solution Absolute Error Tolerance = float]
```

These two flags set the tolerances that are used in calculating the convergence criterion for the update vector in the nonlinear solver. This criterion is

$$\frac{1}{N}\sum_{i=1}^{N}\frac{|\delta_i|}{\varepsilon_R|x_i| + \varepsilon_A} < 1.0, \qquad (3.1)$$

where $\varepsilon_R$ and $\varepsilon_A$ are the relative and absolute tolerances entered in the above input lines, $\delta_i$ is the update for the unknown $x_i$, and $N$ is the total number of unknowns. The quantity on the left side of this inequality is what is output from the solver as the update norm.

The convergence of the nonlinear solver requires that the above inequality be met and that the nonlinear residual drop by two orders of magnitude from its original value. (This ratio is output by the code as the "Ratio of scaled residual_k/residual_0.") Default: $\varepsilon_R = 10^{-3}$ and $\varepsilon_A = 10^{-8}$.

## 3.3.2. Linear Solver Subsection

`[Solution Algorithm =` *string*`]`

This flag chooses the linear solution algorithm from the Aztec package. The choices are listed in Table 3.3. For a description of the different methods, see the Aztec manual [26]. Default = `gmres`.

| Keyword | Linear Solution Algorithm |
|---|---|
| `gmres` | Restarted General Minimized Residual Method |
| `tfqmr` | Transpose-Free Quasi Minimum Residual Method |
| `cg` | Conjugate Gradient Method |
| `cgs` | Conjugate Gradient Squared Method |
| `cgstab` | Stabilized Biconjugate Gradient Method |
| `lu` | Full sparse LU factorization (available only on 1 processor) |

*Table 3.3. This table enumerates the choices of linear* Solution Algorithm *flag. The strings in the left columns are the keywords recognized by MPSalsa.*

`[Convergence Norm =` *integer*`]`

There are five choices for the norm that measures the progress of the linear solver. These are described in Table 3.4. The most common choice is 0, since this corresponds to the norm in the GMRES method. Default = `0`.

`[Preconditioner =` *string*`]`

This flag chooses the preconditioning method. For many problems, a good preconditioner is essential if the linear solver is to converge. The more robust preconditioning methods require more memory. Table 3.5 lists the available options for the preconditioner flag. Default = `no_overlap_ilu`.

| `Convergence Norm` | Specified Norm |
|---|---|
| 0 | $\lVert r^k \rVert_2 / \lVert r^0 \rVert_2$ |
| 1 | $\lVert r^k \rVert_2 / \lVert b \rVert_2$ |
| 2 | $\lVert r^k \rVert_2 / \lVert A \rVert_\infty$ |
| 3 | $\lVert r^k \rVert_\infty / \left( \lVert A \rVert_\infty \lVert x^k \rVert_1 + \lVert b \rVert_\infty \right)$ |
| 4 | $\left( \frac{1}{N} \sum_{i=1}^{N} \left( \frac{r_i^k}{\varepsilon_R \lvert x_i \rvert + \varepsilon_A} \right)^2 \right)^{1/2}$ |

*Table 3.4. The five choices for the* Convergence Norm *flag are shown. The linear system is Ax=b, for which at each iterate k (in the linear solution algorithm),* $r^k = b - Ax^k$*. The tolerances* $\varepsilon_R$ *and* $\varepsilon_A$ *are those used by the nonlinear solver (Eq. (3.1)). For nonlinear problems, an initial guess of* $x^0=0$ *is used, so choices 0 and 1 are equivalent.*

| Keyword | Preconditioner |
|---|---|
| `full_overlap_ilu`<br>`full_overlap_bilu` | ILU(0) and Block-ILU(0) with one level of overlap between processors. |
| `diag_overlap_ilu`<br>`diag_overlap_bilu` | ILU(0) and Block-ILU(0) with overlapping of diagonal blocks between processors. |
| `no_overlap_ilu`<br>`no_overlap_bilu` | ILU(0) and Block-ILU(0) with no overlapping between processors. |
| `poly` | Polynomial preconditioner, with the order specified by the next input line. |
| `sgs` | Domain decomposition, no overlap, symmetric Gauss-Seidel. |
| `jacobi` | Jacobi preconditioner. |
| `none` | No preconditioner applied. |

*Table 3.5. This table enumerates the choices for the* `Preconditioner` *flag. The strings in the left columns are the keywords recognized by MPSalsa. The* `Scaling` *file line has more options that can be used in combination with these.*

`[Polynomial = {LS | NS},` *integer*`]`

When a polynomial preconditioner is selected in the previous input line, this line specifies the type of polynomial and the order. The two choices for the polynomial type are "`LS`" for least-squares, and "`NS`" for Neumann series. The polynomial order is an integer that must be preceded by a comma. For a least-squares polynomial, the choices for the order are 0–9, while for the Neumann series the choice is 0–infinity. Default = `LS,3`.

[Scaling = *string*]

The Scaling option specifies what type of scaling is done by the linear solver at the start of the linear solve. Scaling is similar to preconditioning but is carried out only once at the beginning of the linear solve. Each scaling option may be used in conjunction with any choice of a Preconditioner, although only the symmetric scaling options should be used with the conjugate gradient preconditioner. The available scaling options are listed in Table 3.6. Block Jacobi scaling uses Gaussian elimination to invert the diagonal $(N_u \times N_u)$ blocks of the matrix, where $N_u$ is the number of unknowns per node. The inverted block is then multiplied into the matrix and right-hand side. Row-sum scaling uses a diagonal matrix as the preconditioner, with the row sums as the diagonal entries. Default = row_sum.

| Keyword | Scaling Method |
|---|---|
| block_jacobi | Right hand scaling using the inverted diagonal block. |
| sym_diag | Symmetric (right and left) scaling using the matrix diagonal. |
| row_sum | Right hand scaling with the sum of the absolute values of the column entries. |
| none | No scaling. |

*Table 3.6. This table enumerates the choices for* Scaling. *The strings in the left columns are the keywords recognized by MPSalsa.*

[Orthogonalization = {classical | modified}]

For the GMRES method, the vectors of the Krylov subspace must be made orthonormal. The two options for the Gram-Schmidt orthogonalization method are classical and modified [18]. While the modified method is more stable numerically, its parallel implementation is significantly more costly. In our experience, classical orthogonalization has worked well for the problems we have solved. Default = classical.

[Size of Krylov subspace = *integer*]

For the restarted GMRES method (Solution Algorithm choice gmres), the Krylov subspace size is the number of Krylov vectors to store before restarting. With higher values of this number, convergence of the linear solver is more robust, but more memory is needed. Each directional vector that is saved requires an amount of memory equivalent to an entire solution vector. For finding steady states of large problems, this number can often (and should for maximum efficiency) exceed 100. Default = 64.

[Maximum Linear Solve Iterations = *integer*]

This line specifies the maximum number of iterations allowed in any given linear solve. When this maximum is reached before the residual has been reduced by the specified amount (as specified by the Choice for Inexact Newton Forcing Term and Linear Solver Normalized Residual Tolerance input lines), the linear solver terminates and an error condition is returned to the calling program. For nonlinear problems, the solution is accepted nonetheless and the next Newton step is started. For restarted GMRES, this number is usually picked to be a small integer multiple (2 or 3) of the Krylov subspace size. Default = 300.

[Linear Solver Normalized Residual Tolerance = *float*]

For linear problems and nonlinear problems for which the Choice for Inexact Newton Forcing Term = 4, this input line specifies $\varepsilon_L$, the drop in the residual required by the linear solver before it terminates successfully. The linear residual is checked after every iteration of the linear solver, so the solver does not do more iterations than necessary. Default: $\varepsilon_L = 10^{-4}$ for nonlinear problems; $\varepsilon_L = 10^{-6}$ for linear problems.

## 3.4. Chemistry Specifications

The Chemistry Specifications section of the input file allows control over much of the reaction and diffusion processes for problems with mass transfer. This section is required only if the Problem Type indicates that mass balance equations are to be solved (see Table 3.1). A sample section of the input file, including the Chemistry Specifications header, is shown in Figure 3.4.

```
-----------------------------------------------------------
 Chemistry Specifications
-----------------------------------------------------------
Energy equation source terms        = on
Species equation source terms        = on
Pressure (atmospheres)               = 1.0
Thermal Diffusion                    = off
Multicomponent Transport             = stefan_maxwell
Chemkin file                         = chem.bin
Surface chemkin file                 = surf.bin
Transport chemkin file               = tran.bin
```

*Figure 3.4. Chemistry Specifications section example.*

[Energy equation source terms = {on | off}]

This flag allows the user to turn on and off the energy source terms due to chemical reactions. Default = on.

`[Species equation source terms = {on | off}]`

This flag allows the user to turn `on` and `off` the chemical reactions in the interior of the domain. Surface reactions are controlled separately through the boundary condition section. Default = `on`.

`[Pressure (atmospheres) =` *float*`]`

For problems with a `CHEMKIN` material type (see Section 3.6), the ideal gas equation of state is used to calculate the reaction rates and physical properties, such as density. This flag sets the thermodynamic pressure in the domain, which is assumed to be nearly constant. The local deviation of the pressure due to hydrodynamics, which is captured by the pressure unknown for fluid flow problems, is assumed to be negligible for the low Mach number applications that MPSalsa is written for. This input line is not generally relevant for other material types, although a user could write their own material property functions that use this quantity, which is named `Ptherm` in the code. Default = 1.0.

`[Thermal Diffusion = {on | off}]`

Thermal diffusion -- also called the Soret effect -- can be turned `on` or `off` by this flag. Thermal diffusion can become a significant contributor to mass transfer when gas species of greatly varying molecular weights are exposed to a steep thermal gradient. This flag may be turned off to save computational time when the effect is small, or to simplify the equations for better convergence behavior. The thermal diffusion term can be responsible for a modest increase in time for the matrix fill. Currently, the thermal diffusion term is nonzero only for the `CHEMKIN` material type. Default = `on` for `CHEMKIN` materials.

`[Multicomponent Transport =` *string*`]`

This flag will, in the future, allow the user to switch between different diffusion formulations for multicomponent transport. Currently, mixture-averaged diffusion is the only option, and is specified by the `mixture_avg` flag. Stefan-Maxwell and Dixon-Lewis formulations are planned, and will take the flag values `stefan_maxwell` and `dixon_lewis`. These flags are recognized but not included. Default = `mixture_avg`.

```
[Chemkin file = chem.bin]
[Surface chemkin file = surf.bin]
[Transport chemkin file = tran.bin]
```

These three input lines specify the names of the data files for problems that use Chemkin for the material properties. The Chemkin interpreter program "interp" (see Section 2.3) creates these files with the following names, which are also the defaults: `chem.bin`, `surf.bin`, `tran.bin`.

## 3.5. Enclosure Radiation Specifications

Enclosure radiation algorithms that are used in the CoyoteII code [16] are being included in MPSalsa. However, this capability is still under development and is not yet supported. The input-file section shown in Figure 3.5 may be included; however, it is optional.

```
-------------------------------------------------------------
 Enclosure Radiation Specifications
-------------------------------------------------------------
Enclosure Radiation source terms     = off
```

*Figure 3.5. Enclosure Radiation Specifications section example.*

## 3.6. Material Specifications

In the Material Specifications section of the input file, the user can set the physical properties of the system. The computational domain can consist of multiple materials, each with a unique set of properties; at present, however, the same physics (i.e., governing equations) must be solved in all materials. A multi-physics capability is under development.

An example of the Materials Specifications section is given in Figure 3.6. This section is required by MPSalsa. It differs from the previous sections in that it is mostly free-format. Only the first two lines and the last line are required.

`Number of Materials =` *integer*

This line must be the first line of the Materials Properties section. It specifies the number of materials (usually one) that make up the computational domain. For multiple materials, the input lines that are described below are repeated multiple times. The materials are assigned to a block of elements in the mesh using the `ELEM_BLOCK_IDS` line described below.

The first line for each material specifies the material type, material ID, and material name, and has the format:

`Material_Type =` *integer_id "Material_Name"*

The `Material_Type` string can be one of several keywords. These keywords are listed in Table 3.7. The `CHEMKIN` type is special, in that it tells MPSalsa to get the material properties from the Chemkin database. The *integer_id* is a unique integer identification (ID) number for the material. The user can supply any string, within quotes, as the *Material_Name*, which is only echoed back by MPSalsa in place of the integer ID.

```
-----------------------------------------------------------
      Material ID Specifications
-----------------------------------------------------------
Number of Materials              = 1
BOUSSINESQ                                 = 0    "3Yk-gas"
ELEM_BLOCK_IDS  =  1 2

        NUM_SPECIES      = 3
        SPECIAL_SPECIES_EQN = yes

        SPECIES_NAME     1  Yk_0
        SPECIES_NAME     2  Yk_1
        SPECIES_NAME     3  Yk_2

        DIFF_COEFF       Yk_2  0.4
        DIFF_COEFF       Yk_0  0.5
        DIFF_COEFF       Yk_1  0.6

        WTSPECIES        Yk_0  1.0
        WTSPECIES        Yk_1  1.0
        WTSPECIES        Yk_2  1.0

        DENSITY          =  1.0
        CP               =  2.0
        VISCOSITY        =  3.0
        THERMAL_CONDUCT  =  1.0
        VOL_EXPNS        =  5.0
        G_VECTOR         = 0.0, 9.8, 0.0
        Q_VOLUME_VAR     =  q_xx_yy

        XMF_0            Yk_0  0.2
        XMF_0            Yk_1  0.1
        XMF_0            Yk_2  0.6
        U_INIT = 10.0
        T_INIT = 298.0

END Material ID Specifications
```

*Figure 3.6. Material ID Specifications section example.*

The assignment of the physical and transport properties for the current material follow the `Material_Type` line until they are terminated by the line:

`END Material ID Specifications`

Any entries after this line are ignored.

The material properties can follow in almost any order and all have default values. The only ordering that is required is that the number of species (`NUM_SPECIES`) must be specified before the species names (`SPECIES_NAME`) are given, and that the species names must be given before the species-dependent properties (`DIFF_COEFF`, `WTSPECIES`, `XMF_0`) are specified.

| Material Type | Description |
|---|---|
| SOLID, NEWTONIAN | Usual equations; isotropic conductivity, body force = $\rho g$. |
| BOUSSINESQ | Body force term replaced by linear Boussinesq approx. in Temperature. |
| CHEMKIN | All physical and transport properties calculated from Chemkin --ideal gas equation of state. Properties vary with thermodynamic state. |

*Table 3.7. List of* `Material_Type` *designators recognized by MPSalsa.*

| Keyword | Argument | Default | Description |
|---|---|---|---|
| ELEM_BLOCK_IDS | *integer* list | | List of element blocks, as specified by the mesh generator, that compose the current material. |
| G_VECTOR | 3 *float* | 0, 0, 0 | The *x*-, *y*-, and *z*-components of the gravity vector. The units are arbitrary except for CHEMKIN materials, where cgs units are the default. |

*Table 3.8. General Keywords: first of four tables listing and describing keywords recognized for the specification of material properties.*

| Keyword | Argument | Default | Description |
|---|---|---|---|
| DENSITY | *float* or VARIABLE_PROP | 1.0 | A floating-point argument sets a constant density value; the VARIABLE_PROP flag tells MPSalsa to get the value from the function "user_density." |
| VISCOSITY | *float* or VARIABLE_PROP | 1.0 | A floating-point argument sets a constant viscosity value; the VARIABLE_PROP flag tells MPSalsa to get the value from the function "user_viscosity." |
| CP | *float* or VARIABLE_PROP | 1.0 | A floating-point argument sets a constant heat capacity; the VARIABLE_PROP flag tells MPSalsa to get the value from the function "user_Cp." |
| THERMAL_CONDUCT | *float* or VARIABLE_PROP | 1.0 | A floating-point argument sets a constant thermal conductivity; the VARIABLE_PROP flag tells MPSalsa to get the value from the function "user_cond." |
| VOL_EXPNS | *float* | 0.0 | Volumetric expansion coefficient (units are inverse temperature); used only for BOUSSINESQ materials. |
| T_NAUGHT | *float* | 0.0 | Reference temperature for BOUSSINESQ approximations. |
| Q_VOLUME | *float* | | Constant volumetric source added to the heat balance. |
| Q_VOLUME_VAR | *fn_name* | | A volumetric source computed by the function *fn_name* and added to the heat balance. |
| VISC_DISSP | | | Causes viscous dissipation terms to be added to the heat balance; this flag is not currently implemented. |

*Table 3.9. Fluid and Thermal Properties: second of four tables listing and describing keywords recognized for the specification of material properties.*

| Keyword | Argument | Default | Description |
|---|---|---|---|
| NUM_SPECIES | *integer* | 0 | Number of species for problems that include mass transfer. |
| SPECIES_NAME | *integer, string* | | The integer ID of the species, between 1 and the entry for NUM_SPECIES, followed by the name of the species. |
| WTSPECIES | *string, float* | | The molecular weight of species *string*, where *string* is a SPECIES_NAME input above. WTSPECIES should be given for each species. |
| DIFF_COEFF | *string, float* | 1.0 | The diffusion coefficient of species *string*, where *string* is a SPECIES_NAME input above. DIFF_COEFF should be given for each species. |
| SPEC_SPECIES_EQN | {yes \| no} | yes for CHEMKIN materials; no, otherwise. | When this flag is yes, the last species equation is replaced by the requirement that the sum of the mass fractions is one. For CHEMKIN material types, the default value of yes may not be overridden. |
| Y_VOLUME | *float* | | A constant volumetric source term that is the same for all species. |
| Y_VOLUME_VAR | *fn_name*, {SINGLE \| MULTIPLE} | | Volumetric source term for each mass balance computed by the user-specified function *fn_name*. SINGLE or MULTIPLE indicates whether the function returns one equation's source term at a time or the entire vector of source terms at once. |
| JACOBIAN_SRC_TERMS_VAR | *fn_name* | | If this string is present, the function *fn_name* is used to compute the Jacobian entries due to the source terms; otherwise, a numerical Jacobian is computed. |

*Table 3.10. Mass Transfer Properties: third of four tables listing and describing keywords recognized for the specification of material properties.*

The recognized strings (or keywords) that can be used to specify material properties are listed and described in Table 3.8, Table 3.9, Table 3.10, and Table 3.11. The strings are organized into separate tables only for this document; there are no distinctions in the code.

The ELEMENT_BLOCK_IDS line in Table 3.8 is required for each material type. All element blocks in the computational domain (see discussion in Section 2.1) must be specified in one and only one material-type section.

For CHEMKIN material types, the number of species and their names are specified in the Chemkin linking files. Additionally, the molecular weights, diffusion coefficients, mixture viscosity, mixture heat capacity, mixture thermal conductivity, multicomponent diffusion

| Keyword | Argument | Description |
|---|---|---|
| U_INIT | *float* | The initial value for the *x*-component of all the velocity unknowns. |
| V_INIT | *float* | The initial value for the *y*-component of all the velocity unknowns. |
| W_INIT | *float* | The initial value for the *z*-component of all the velocity unknowns. |
| P_INIT | *float* | The initial value for all of the pressure unknowns. |
| T_INIT | *float* | The initial value for all of the temperature unknowns. |
| XMF_0 | *string, float* | The initial species mole fractions, which are translated to mass fractions and assigned to the mass-fraction unknowns. The *string* is the name of the species, which comes from the SPECIES_NAME line or the Chemkin data file. |

*Table 3.11. Initial Value Specifications: fourth of four tables listing and describing keywords recognized for the specification of material properties.*

coefficients, mixture density, and volume expansion coefficient are all specified or calculated from Chemkin functions. It is an error to redefine them in for a CHEMKIN material.

## 3.7. Boundary Condition Specifications

Generalized surface vectors and boundary conditions for a problem are specified in the Boundary Condition section of the input file. This section is required by MPSalsa. An example for a WHOLE_ENCHILADA problem is given in Figure 3.7.

### 3.7.1. Generalized Surfaces

A generalized surface is a side set in the ExodusII file for which the outward normal and tangential vectors of the corresponding geometric surface are given in the input file. These vectors may be used to specify side-set boundary conditions in the surface's normal and tangential directions. The number of generalized surfaces included in the input file is listed first.

```
    Number of generalized surfaces = integer
```

The format for specifying each generalized surface follows.

```
    GENERALIZED_SURFACE side_set_number number_of_vectors
          TANGENT    {real real [real] | function_name}
          [TANGENT   {real real [real] | function_name}]
          [NORMAL    {real real [real] | function_name}]
```

where

*side_set_number* = the side set ID number in ExodusII, and

```
 -----------------------------------------------------------------
  Boundary Condition Specifications
 -----------------------------------------------------------------
 Number of Generalized Surfaces = 2
 GENERALIZED_SURFACE 4 2
        TANGENT  0.8 0.6 0.
        TANGENT -0.6 0.8 0.
 GENERALIZED_SURFACE 5 3
        NORMAL user_normal
        TANGENT user_tangent1
        TANGENT 0. 0. 1.

 Number of BC = 12
 BC = T_BC DIRICHLET SS 1 INDEPENDENT 300. 0
 BC = T_BC NEUMANN SS 5 INDEPENDENT f_xx_yy 1
        BC_DATA = 1.0 2.0 0.5
 BC = T_BC MIXED SS 4 DEPENDENT jbc_fn 0.5 0.1 0.2 f_fn 0
 BC = P_BC DIRICHLET NS 9 INDEPENDENT 1. 0

 BC = U_BC DIRICHLET SS 1 INDEPENDENT 0. 0
 BC = VEL_TAN1_BC DIRICHLET GS 1 INDEPENDENT f_xy_spin_disk 1
        BC_DATA = 100.0 0. 0.

 BC = V_BC DIRICHLET SS 1 INDEPENDENT 0. 0
 BC = VEL_TAN2_BC DIRICHLET GS 1 DEPENDENT f_xy_spin_disk 1
        BC_DATA = 100.0 0. 0.

 BC = W_BC DIRICHLET SS 1 INDEPENDENT -9. 0
 BC = VEL_NORM_BC DIRICHLET GS 1 DEPENDENT surface_chemkin_bc 0

 BC = Y_BC DIRICHLET SS 1 INDEPENDENT f_mole_fraction 1
        SPECIES_LIST = 2 1 4 3
        BC_DATA = 1.232900e-04 1.095458e-02 9.889221e-01 0.0
 BC = Y_BC DIRICHLET SS 4 DEPENDENT surface_chemkin_bc 0
        SPECIES_LIST = ALL
```

*Figure 3.7. Example of the Boundary Condition Specification section of the input file.*

*number_of_vectors* = the number of vectors used to describe the surface.

Two orthogonal unit tangent vectors should be given for 3-D problems; one unit tangent vector suffices for 2-D problems. The unit outward-normal vector is optional; for boundary conditions in the outward normal direction, MPSalsa uses a vector normal to the mesh geometry if a vector normal to the surface is not specified.

The outward normal vector and tangent vectors on the surface are described on the following line(s). Either the coordinates of a vector or the name of a function returning the vector may be used to specify the vectors (see Section 4.3). The example in Figure 3.7 includes two generalized surfaces. The first consists of side set 4 with two unit tangent vectors; since a normal vector is not specified, outward normal vectors on the surface are computed within MPSalsa. The second consists of side set 5 with the outward normal vector returned by `user_normal`, a tangent vector returned by `user_tangent1`, and a constant tangent vector.

MPSalsa numbers the generalized surfaces (starting from one) in the order they appear in the input file. Boundary condition statements for generalized surfaces reference the generalized surface number assigned by MPSalsa as their *set_id* (see Section 3.7.2). Alternatively, the number of the side set for which the generalized surface is described can be specified; MPSalsa associates the appropriate generalized-surface definition with the side set.

### 3.7.2. Boundary Conditions

The number of boundary conditions included in the input file is specified before the boundary conditions are listed:

`Number of BC =` *integer*

Each boundary condition has the following format:

`BC =` *bc_name bc_type set_type set_id dependence_flag bc_values num_data_lines*

where

*bc_name* = {`U_BC | V_BC | W_BC | T_BC | P_BC | Y_BC | VEL_NORM_BC | VEL_TAN1_BC | VEL_TAN2_BC`};

*bc_type* = {`DIRICHLET | NEUMANN | MIXED`};

*set_type* = {`NS | SS | GS`}

*set_id* = side set ID number, node set ID number, or generalized surface number;

*dependence_flag* = {`DEPENDENT | INDEPENDENT`};

*bc_values* is described in Table 3.13; and

*num_data_lines* = *integer*.

The *bc_name* indicates the variable to which the boundary condition should be applied. Possible values for *bc_name* are listed in Table 3.12. All velocity boundary conditions on a side set must be specified in the same coordinate system; normal and tangential velocity boundary conditions (`VEL_NORM_BC`, `VEL_TAN1_BC`, `VEL_TAN2_BC`) may not be used with `U_BC`, `V_BC`, or `W_BC` on the same side set.

| *bc_name* | Variable to which the boundary condition is applied. |
|---|---|
| U_BC | velocity in the *x*-direction. |
| V_BC | velocity in the *y*-direction. |
| W_BC | velocity in the *z*-direction. |
| T_BC | temperature. |
| P_BC | pressure. |
| Y_BC | mass fractions. |
| VEL_NORM_BC | velocity in the direction normal to the surface. Note: only Dirichlet BCs are valid for VEL_NORM_BC. |
| VEL_TAN1_BC | velocity in the direction of the first tangent vector (given by a generalized surface) to the surface. Note: only Dirichlet BCs are valid for VEL_TAN1_BC. |
| VEL_TAN2_BC | velocity in the direction of the second tangent vector (given by a generalized surface) to the surface. Note: only Dirichlet BCs are valid for VEL_TAN2_BC. |

*Table 3.12. Boundary condition names and their corresponding variables.*

The *bc_type* indicates the type of boundary condition to apply. Three types of boundary conditions are implemented in MPSalsa: Dirichlet, Neumann, and Mixed (Robin). Dirichlet boundary conditions have the following forms:

$$y = f(t, \mathbf{x}, \mathbf{u}, P, T, \mathbf{Y}) \text{ for U\_BC, V\_BC, W\_BC, P\_BC, T\_BC or Y\_BC,} \tag{3.2}$$

$$\mathbf{n} \bullet \mathbf{u} = f(t, \mathbf{x}, \mathbf{u}, P, T, \mathbf{Y}) \text{ for VEL\_NORM\_BC, and} \tag{3.3}$$

$$\mathbf{t} \bullet \mathbf{u} = f(t, \mathbf{x}, \mathbf{u}, P, T, \mathbf{Y}) \text{ for VEL\_TAN1\_BC and VEL\_TAN2\_BC,} \tag{3.4}$$

where $y = u_1, u_2, u_3, P, T,$ or $\mathbf{Y}$ is the unknown whose boundary condition is assigned, $\mathbf{n}$ and $\mathbf{t}$ are unit outward-normal and tangential vectors specified in a generalized-surface definition or computed by MPSalsa, and $f$ is a function of time $t$, position $\mathbf{x}$, and the solution variables $\mathbf{u}$, $P$, $T$, and $\mathbf{Y}$ at $\mathbf{x}$.

Neumann boundary conditions take the form

$$\mathbf{n} \cdot \mathbf{q}_c = f(t, \mathbf{x}, \mathbf{u}, P, T, \mathbf{Y}) \text{ , } \mathbf{q}_c = -\lambda \nabla T \text{, for the temperature equation,} \tag{3.5}$$

$$\mathbf{n} \cdot \mathbf{j}_k = f(t, \mathbf{x}, \mathbf{u}, P, T, \mathbf{Y}) \text{ , } \mathbf{j}_k = \rho Y_k \mathbf{V}_k \text{, for the } k^{th} \text{ mass fraction equation, and} \tag{3.6}$$

$$(\mathbf{T}\mathbf{n})_l = f(t, \mathbf{x}, \mathbf{u}, P, T, \mathbf{Y}) \text{ for the } l^{th} \text{ momentum equation,} \tag{3.7}$$

where $\mathbf{n}$ is the unit outward normal vector, $\lambda$ is the mixture thermal conductivity, $\rho$ is the mixture density, $\mathbf{V}_k$ is the diffusion velocity of species $k$, and $\mathbf{T}$ is the shear stress tensor.

Mixed boundary conditions replace the function $f$ on the right-hand side of (3.5)-(3.7) with

$$h\,(y-y_0)+af(t,\mathbf{x},\mathbf{u},P,T,\mathbf{Y}), \tag{3.8}$$

where $a$ is a floating-point constant, and $h = h(t, \mathbf{x}, \mathbf{u}, P, T, \mathbf{Y})$ and $y_0 = y_0(t, \mathbf{x}, \mathbf{u}, P, T, \mathbf{Y})$ are functions of time $t$, position $\mathbf{x}$, and the solution vector at $\mathbf{x}$.

In MPSalsa, `DIRICHLET` boundary conditions replace the finite-element equation for an unknown. `NEUMANN` and `MIXED` boundary conditions add a surface integral contribution to the finite-element equation for an unknown. Only `DIRICHLET` boundary conditions are currently implemented for `VEL_NORM_BC`, `VEL_TAN1_BC`, and `VEL_TAN2_BC`. `NEUMANN` and `MIXED` types will be added for these boundary conditions in the future. Pressure boundary conditions (`P_BC`) may also be only of type `DIRICHLET`. All other boundary conditions may be of any type.

The ExodusII side or node set to which the boundary condition is applied is specified by a *set_type* and the *set_id_num*. The *set_type* is `SS` for a side set, `NS` for a node set, or `GS` for a generalized surface side set. The *set_id_num* is the number of the side or node set in the ExodusII file, or the number of the generalized surface defined in the input file. `NEUMANN` and `MIXED` boundary conditions may be applied only to side sets or generalized surfaces; `DIRICHLET` boundary conditions may be applied to node sets, side sets, or generalized surfaces.

Boundary condition functions $f$, $h$, and $y_0$ in (3.2) - (3.8) may depend on the solution. If terms resulting from this dependence are to be included in the Jacobian matrix, the *dependence_flag* should be set to `DEPENDENT`; otherwise, the *dependence_flag* should be set to `INDEPENDENT`. Mixed boundary conditions should be labeled `DEPENDENT` only if at least one of the functions $f$, $h$, or $y_0$ depends on the solution. For `INDEPENDENT` mixed boundary conditions, the analytic Jacobian contribution

$$\frac{\partial}{\partial y}(h(t,\mathbf{x})\,[y-y_0(t,\mathbf{x})]+af(t,\mathbf{x})) = h(t,\mathbf{x})$$

is computed by MPSalsa and included in the Jacobian.

The *bc_values* vary depending on *bc_type* and *dependence_flag*; the correct combinations of arguments are listed in Table 3.13. The values of $f$, $h$, and $y_0$ in (3.2) - (3.8) may be given by a real number or a function. The value of $a$ in (3.8) is a real number. Analytic Jacobian entries may be given for `DEPENDENT` boundary conditions through specification of a *jacobian_function_name*, a function that returns the partial derivative of the boundary condition with respect to the solution unknowns. If no *jacobian_function_name* is specified, a numerical Jacobian is used for DEPENDENT boundary conditions. Many functions for $f$, $h$, and $y_0$ and their analytic Jacobian entries are included in MPSalsa; see Section 4.2 and Appendix A.1.

| *bc_type* | *dependence_flag* | *bc_values* |
|---|---|---|
| DIRICHLET | INDEPENDENT | {*f_function_name* \| *f_real*} |
| DIRICHLET | DEPENDENT | [*jacobian_function_name*] {*f_function_name* \| *f_real*} |
| NEUMANN | INDEPENDENT | {*f_function_name* \| *f_real*} |
| NEUMANN | DEPENDENT | [*jacobian_function_name*] {*f_function_name* \| *f_real*} |
| MIXED | INDEPENDENT | {*h_function_name* \| *h_real*} {*y0_function_name* \| *y0_real*} {*a_real*} {*f_function_name* \| *f_real*} |
| MIXED | DEPENDENT | [*jacobian_function_name*] {*h_function_name* \| *h_real*} {*y0_function_name* \| *y0_real*} {*a_real*} {*f_function_name* \| *f_real*} |

*Table 3.13. Boundary condition specification of bc_values for various bc_types and dependence_flags.*

Additional data may be passed to boundary condition functions through the use of BC_DATA lines. The number of these lines for a boundary condition is given as the last entry, *num_data_lines,* on the BC line. BC_DATA lines are formatted as follows:

BC_DATA = *data_type data_values*

where

*data_type* = {FLOAT | INT | INTEGER | FUNCTION}; and

*data_values* = a list of real numbers (for *data_type* FLOAT), integers (for *data_types* INT and INTEGER), or function names (for *data_type* FUNCTION). These data values are stored in one-dimensional arrays associated with the boundary conditions and may be accessed by user-defined functions. See Section 4.2.1 for examples of the use of these values.

Examples of each type of boundary condition are included in Figure 3.7. A few examples are detailed below.

```
BC = P_BC DIRICHLET NS 9 INDEPENDENT 1. 0
```

A Dirichlet boundary condition value of 1 is applied to pressure unknowns in node set 9.

```
BC = VEL_NORM_BC DIRICHLET GS 1 DEPENDENT surface_chemkin_bc 0
```

A Dirichlet outward-normal velocity boundary condition is applied to velocity unknowns on the first generalized surface listed in the input file. The value of the boundary condition is computed in function `surface_chemkin_bc` (see Appendix A.1.1). Since the boundary condition is `DEPENDENT` but no analytic Jacobian function is specified, numerical Jacobian entries for the boundary condition are computed.

```
BC = T_BC NEUMANN SS 5 INDEPENDENT f_xx_yy 1
     BC_DATA = 1.0 2.0 0.5
```

A Neumann boundary condition is applied to the temperature equations for nodes in side set 5. The value of the boundary condition is computed in function `f_xx_yy`. No Jacobian entries for the boundary condition are generated since the boundary condition is `INDEPENDENT`. `BC_DATA` values of 1.0, 2.0, and 0.5 are passed to function `f_xx_yy` for use in computing the boundary condition value.

```
BC = T_BC MIXED SS 4 DEPENDENT jbc_fn 0.5 0.1 0.2 f_fn 0
```

A Mixed boundary condition of the form

$$\mathbf{n} \cdot \mathbf{q}_c = 0.5\,(T - 0.1) + 0.2\ \text{f\_fn}\,(t, \mathbf{x}, \mathbf{u}, P, T, \mathbf{Y})$$

is applied to the temperature equations for nodes in side set 4. The boundary condition is `DEPENDENT`; function `jbc_fn`$(t, \mathbf{x}, \mathbf{u}, P, T, \mathbf{Y})$ is called to compute analytic Jacobian entries for the boundary condition terms.

Default: If no boundary condition is specified for an unknown in a node- or side-set, a natural boundary condition with value 0 is applied to the equation for the unknown. Thus, the default boundary condition for temperature, mass fractions or velocities is effectively `NEUMANN` with $f(t, \mathbf{x}, \mathbf{u}, P, T, \mathbf{Y}) = 0$ in (3.5), (3.6), or (3.7), respectively.

### 3.7.2.1. Mass Fraction Boundary Conditions

A mass fraction boundary condition (`Y_BC`) may be applied to one, some or all of the species unknowns in the node or side set. The `SPECIES_LIST` input line indicates to which species the boundary condition should be applied. This line must directly follow the `BC` statement.

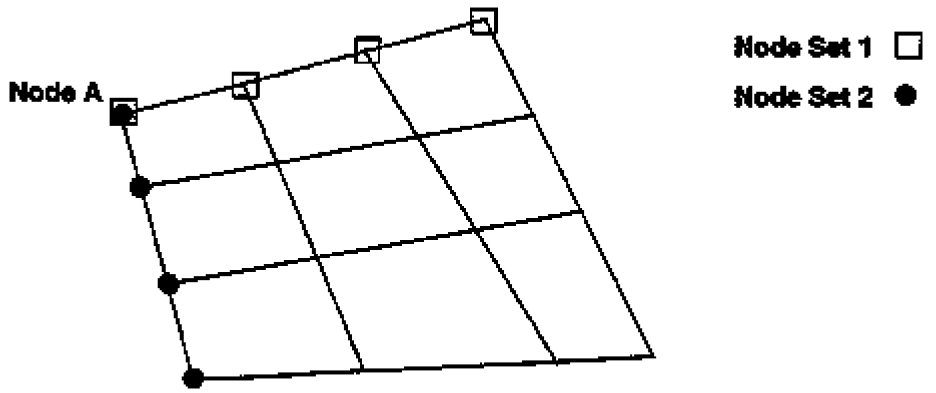
`SPECIES_LIST = {ALL` | *list of species numbers* | *list of species names*`}`

The keyword ALL states the boundary condition should be applied to all species in the problem. Individual species may be listed by number or name, where the name is given either in the Materials Specifications (see Section 3.6) or the Chemkin files.

All Y_BC boundary conditions are specified in terms of mass fractions rather than mole fractions. DIRICHLET boundary conditions may also be specified as mole fractions via the function f_mole_fraction included in MPSalsa (see Section A.1.4).

### 3.7.2.2. Precedence of Boundary Conditions

For unknowns at nodes where two or more side or node sets intersect, Dirichlet boundary conditions always have precedence over other types of boundary conditions. That is, if a node has unknowns upon which Dirichlet and, say, Neumann boundary conditions are specified, the Dirichlet boundary condition is the boundary condition imposed. Moreover, the first Dirichlet boundary condition in the input file for such an unknown is the one applied. If a node belongs to more than one node or side set, as Node A does in Figure 3.8, the first Dirichlet boundary condition for each unknown at that node is the one applied. In Figure 3.8, the Dirichlet boundary condition for node set 2 would be applied to Node A.



```
BC = T_BC DIRICHLET NS 2 INDEPENDENT 300. 0
BC = T_BC DIRICHLET NS 1 INDEPENDENT 100. 0
```

*Figure 3.8. Example demonstrating the precedence of Dirichlet boundary conditions. Node A belongs to both node set 1 and node set 2. Its temperature would be set to a value of 300 in this example.*

## 3.8. Initial Condition/Guess Specifications

In the Initial Condition/Guess Specifications section of the input file, users can specify what type of initial guess or initial conditions to use. This section is required by MPSalsa. An example is shown in Figure 3.9. MPSalsa's initial guess for the solution vector is established in several steps. The first step involves preprocessing the solution vector by setting all solution components to a value of zero. Next the Set Initial Condition/Guess line described below is processed. Then, if the solution is not being read from an ExodusII file, all solution variables are set to their "INIT" values specified in the Material Specifications section of the input file, if any are specified. (For example, this is where the condition that the sum of the mass fractions must equal one is enforced for CHEMKIN material types.) Finally, an additional user-supplied function may be invoked as the last step. The remainder of this section describes each of the lines in the Initial Condition/Guess Specifications section of MPSalsa's input file.

```
----------------------------------------------------------
Initial Condition/Guess Specifications
----------------------------------------------------------
Set Initial Condition/Guess              = constant 0.0
Apply Function                           = no
Time Index to Restart From               = 1
```

*Figure 3.9. Example of Initial Condition/Guess Specifications section of the input file.*

`[Set Initial Condition/Guess =` *string* `[`*value*`]]`

This line is used to specify how to initialize the solution vector after the initial default processing is carried out. Valid options for this line are listed below:

= `constant [`*value*`]`

This option initializes all components of the solution vector that do not have material defaults to the constant value *value*. Default: *value* = 0.

= `random`

This option randomly assigns initial solution vector values in the interval [0,1].

= `exoII_file`

Previously stored solution values in the Output FEM file, named in the General Specification Section, are used as initial values. This option is used for restarts.

Default = `constant 0`.

`[Apply function = {`*function name* `| no}]`

A user-written *function* can be specified on this line to process the initial guess. This function is executed after the `Set Initial Condition/Guess` input line so the function can be dependent on a solution read in from an ExodusII file. See Section 4.4 for details on how to write this function. Default = `no`.

`[Time Index to Restart From =` *integer*`]`

This line specifies the index of the time step from which to perform restarts or take the initial guess. This parameter is only pertinent if the `Set Initial Condition/Guess` value is `exoII_file`. Restarts can be performed from any data on the same geometry for steady or time-varying problems. Default = 1 if `Initial Guess = exoII_file`; ignored otherwise.

## 3.9. Output Specifications

In the Output Specifications section, the user may specify how output is to be performed to the ExodusII results file. Items such as which variables to output, how often to output these variables, and whether or not a user-definable subroutine is called are specified in this section. An example of this section is given in Figure 3.10. This section is optional; if it is absent, no output will be performed. A detailed description of each of the lines in the Output Specifications section follows.

```
------------------------------------------------------------
 Output Specifications
------------------------------------------------------------
User Defined Output              = no
Parallel Output                  = no
Scalar Output                    = yes
Time Index to Output To          = 1
Nodal variable output times:
       every 1 steps
Number of nodal output variables= 1
Nodal variable names:
       Temperature
Number of global output variables= 1
Global variable names:
       Delta_time

Test Exact Solution Flag                          = 0
Name of Exact Solution Function                   = f_xx_yy
```

*Figure 3.10. Example of Output Specifications section in the input file.*

`[User Defined Output = {yes | no}]`

This flag indicates whether the standard user-defined function, `user_out`, should be called to output information to *stdout*. This routine allows user-customized output to be added easily. The routine currently distributed in MPSalsa prints out the maximum, minimum, and average value of each unknown as well as the positions of the maximum and minimum. Default = `yes`.

`[Parallel Output = {yes | no}]`

This option allows the user to specify whether or not parallel output should be performed. It can be used simultaneously with scalar output. See Section 2.5 and Section 3.10 for more information on parallel I/O. Default = `no`.

`[Scalar Output = {yes | no}]`

This option allows the user to specify whether or not output to a scalar ExodusII results file should be performed. The name of the file is specified in the General Problem Specifications section (see Section 3.1). Default = `no`.

`[Time Index to Output To =` *integer*`]`

This line is needed only when (1) the MPSalsa run is a restart, and (2) the user wishes to control where in the ExodusII output file (which was used as the restart input file) the output is written. If the line is absent and the run is a restart, new output is appended to the end of the ExodusII output/restart file. When this line included under these conditions, it specifies at what time index (in the restart file) the output should start. The restart file will be overwritten at the time index specified. Note that the initial guess, as read during restarts, is output first. It is therefore suggested that the value of `Time Index to Output To` be set equal to the `Time Index to Restart From` (see Section 3.8) so as to preclude having the same set of values stored twice in the file. Default = output appended to the end of the ExodusII output file.

`[Nodal variable output times:]`
*string*

This line specifies how often during transient runs output of the variables is to be performed. Valid values for *string* are

`every` *n* `steps` -- where *n* is a positive integer

or

`every` *x.xx* `{seconds|units|mins}` -- where *x.xx* is a real positive number.

Several things should be noted about this line. (1) The units are currently ignored since there is no way to specify what these units are for time stepping; (2) the variables to be output are named in the `Nodal variable names` line in the Output Specifications section; and (3) outputting `every` *x* `{seconds|units|mins}` outputs when the time value is the first time value greater than $n*x$, for any integer *n*. Similarly, the next time step output will be the first to have a time value greater than $(n+1)*x$. Default = output every time step.

`[Number of nodal output variables =` *integer*`]`

The number of nodal variables to output is specified here. Default = the total number of variables.

`[Nodal variable names:]`

*string1*
*string2*
.
.
.
*stringN*

The names of the nodal variables to output are given here. The number of nodal variable names *N* is given in the `Number of nodal output variables` line. Valid variable names are

```
Temperature
Velocity
Pressure
Mass_Fraction
Displacement
```

where any combination of the above is valid. The keyword `List` is supported for the variable name `Mass_Fraction`. If the name is followed, on the same line, by the word `List`, a list of species names is expected to follow until the keyword `endlist` is found. For example:

```
Mass_Fraction List
     SIF4, H2, H
     N2, N
     SIHF3
endlist
```

The case of the keywords is not significant. Default: all nodal variables are written in the default order.

`[Number of global output variables =` *integer*`]`

This line is used to specify the number of global variables that are to be output to the ExodusII results file. Global variables are single-valued variables that only have the single dimension of time. They are used to store parameters, timing information, global solution information, etc. Default = 0.

`[Global variable names:]`

*string1*
*string2*

.
.
.

*stringN*

The names of the global variables to be output to the ExodusII results file is given here. The number of global variables $N$ to output is specified in the line `Number of global output variables`. Examples of variable names are

```
Time_index
Delta_time
Matrix_Fill_Time
Matrix_Solve_Time
```

This line is required only if the number of global variables to output is greater than zero. The variable names are case-insensitive. In the future, we hope to allow the user to define additional global variables on this line. The pre-processor "guacamole" will install space for them in the output file, and the routine `user_out` will be used to output values for these variables during an MPSalsa run. Default = none.

`[Test Exact Solution Flag = {0 | 1} [SUMMARY]]`

This line specifies whether or not the computed solution should be tested against a known analytic solution; 0 = off, 1 = on. This comparison includes $L^2$-norm and max-norm error computations. Additional information on the location of the maximum error and an estimate of the largest characteristic length of an element in the FE mesh is provided. The optional keyword SUMMARY will lead to a separate error analysis for each variable in addition to the entire solution vector. Default = 0.

`[Name of Exact Solution Function =` *string*`]`

This line gives the name of the function that will be called to evaluate the accuracy of the computed solution. The generic function `user_bc_exact` may be used by programming the desired exact solution function in the file "rf_user_bc_exact_fn.c." Default = none.

## 3.10. Parallel I/O Specifications

The Parallel I/O Section is used to specify characteristics about parallel disk subsystems connected to specific machines. This section of the input file is optional; if it is absent, no parallel I/O will be performed. An example is given in Figure 3.11. This section of the input file also contains subsections for different parallel architectures. These subsections can remain in the file with the user specifying which architecture to use at run time. In this manner the file can be set up for a number of architectures (currently nCUBE and Intel Paragon) without rewriting the section each time a run is performed on a different architecture.

```
-------------------------------------------------------------------
              Parallel I/O section
-------------------------------------------------------------------
Machine             = paragon
Staged writes       = yes

------------------
ncube subsection
------------------
Number of controllers= 8
Disks per controller= 1
Root location       = //df
Subdirectory        = jns/fireset
Offset numbering from zero= 0

-------------------
paragon subsection
-------------------
Number of RAID controllers= 48
Root location       = /raid/io_
Subdirectory        = tmp/jns/fireset
Offset numbering from zero= 1
```

*Figure 3.11. Example Parallel I/O section.*

`[Machine =` *string*`]`

This line is used to specify the computer architecture. Currently supported architectures are `paragon`, and `ncube`. Default = `paragon`.

`[Staged writes = {yes | no}]`

This lines specifies whether or not writes to parallel disks should be staged. With staging, only one processor writes to each disk at a time. Staging avoids problems with temporary file name conflicts and limits on the number of concurrent open files on a single disk. It is recommended that staging be set to `yes`. Default = `yes`.

`[Number of controllers =` *integer*`]`

This line is specific to the nCUBE subsection and indicates how many controllers should be used in performing the I/O. It must be less than or equal to the number of disk controllers that are actually attached to nCUBE. Default = none; error when not specified for parallel I/O on the nCUBE.

`[Disks per controller =` *integer*`]`

This line is specific to the nCUBE subsection and indicates how many of the disks attached to each of the controllers should be used to perform the I/O. It should be less than or equal to the number of actual disks attached to each controller. Default = none; error when not specified for parallel I/O on the nCUBE.

`[Number of RAID controllers =` *integer*`]`

This line is specific to the Intel Paragon and indicates how many RAID controllers should be used to perform the I/O. It must be less than or equal to the actual number of controllers on the machine. The number of RAID disks is equal to the number of RAID controllers on an Intel Paragon system. Default = none; error when not specified for parallel I/O on the Paragon.

`[Root location =` *string*`]`

The root location is the root directory where writes to the parallel disk subsystem are to be performed. Generally, parallel disk subsystems are in directories that begin with a string. Embedded in the last part of the string is an integer identifying a particular disk. On an nCUBE system, for example, //df00 would be used to write to the first controller and first disk attached to that controller. Similarly, for an Intel Paragon, the user could access the first disk by writing to /pfs/io_01 and the second disk by writing to /pfs/io_02. The value to be specified on the `Root Location` line of the input file is the full pathname of the disk device excluding the identifying integer ID. Figure 3.11 shows examples of the value of `Root Location` for each of these cases. Default = none; error when not specified for parallel I/O.

`[Subdirectory =` *string*`]`

The `Subdirectory` line specifies the subdirectory on the parallel file system in which MPSalsa should look for parallel output and input files. It should not begin with a "/" character. Default = none; error when not specified for parallel I/O.

`[Offset numbering from zero =` *integer*`]`

The offset numbering specifies on which parallel disk I/O should begin. For example, if MPSalsa is to be run on an Intel Paragon using 16 RAIDs beginning with /raid/io_08 then the value of the offset should be set to 8. Default = 1.

## 3.11. Function Data Specifications

Users may pass problem-specific data to functions using the Function Data Specification section of the input file. The Function Data section is optional; users need not include it in the input file if they do not need problem-specific data. An example of the Function Data section is included in Figure 3.12. The functions are used for boundary conditions, material properties, specialized solution output, volumetric source terms, and testing of the code against exact solutions. Four types of data may be passed to functions: integers, reals, strings, and tables.

```
-----------------------------------------------------------
 Data Specification for User's Functions
-----------------------------------------------------------
Number of functions to pass data to = 2

Function = user_bc_exact 4
FN_DATA = -100. -200. -300. -400.
FN_DATA = FLOAT -500. -600.
FN_DATA = STRING VELOCITY APPLICATIONS CZAR
FN_DATA = INT -1 -2 -3 -4 -5

Function = lookup_table_1 2
FN_DATA = STRING TEMPERATURE
FN_DATA = TABLE 6 2
       0      32
       20     68
       40     104
       60     140
       80     176
       100    212
```

*Figure 3.12. An example of the Function Data Specification section of the input file.*

The number of functions that use function data is specified first, with default = 0. For each function, the function name and the number of FN_DATA lines to be passed to it are listed.

> [Number of functions to pass data to = *number of functions*]
> Function = *function_name num_data_lines*

Each FN_DATA line consists of the type of data (INT, FLOAT, STRING, or TABLE). The default is FLOAT. For INT, FLOAT, and STRING data, the data then follows the type keyword. A FLOAT is stored as a double-precision number. Each STRING may be up to 32 characters long.

> FN_DATA = [FLOAT | INT | STRING] *list of data*

TABLES allow the user to supply tabular data to a function. The dimensions of the table follow the TABLE keyword:

```
FN_DATA = TABLE #rows_in_table #columns_in_table
```

The `TABLE` data are included on the lines following the `FN_DATA = TABLE` line. Only one table may be specified in each entry for a function.

Several functions that require function data are included in MPSalsa. Examples are `time_history_line`, which writes to a file the solution along a line in the domain, `time_history_points`, which writes to a file the solution at a set of points in the domain, and look-up table functions `lookup_table_1` and `lookup_table_2`, which interpolate data using a `TABLE` from the function data section of the input file. These and other functions that require user-defined function data are described in Section 4 and Appendix A.

# 4. User Functions

Many features in MPSalsa can be adapted for specific applications through user functions. These functions provide the greatest flexibility for users to control their own simulations. User functions are already included in MPSalsa for quantities such as variable material properties, boundary conditions, and solution measures; users must change only the computations in these routines to calculate the properties for their problems. For some quantities, such as boundary conditions and source terms, users can also write their own functions and compile them into MPSalsa. This process, however, requires more effort and code modification than using the included user functions. This chapter describes the various user functions available and their usage in MPSalsa and the input file. For applicable properties, instructions for including new functions in MPSalsa are also given. For all functions, the units are arbitrary except for `CHEMKIN` materials for which cgs units are the default.

MPSalsa is written in the "C" programming language. The following discussion of modifications to MPSalsa's user functions assumes the user has some knowledge of "C."

## 4.1. Material Properties

### 4.1.1. Heat Capacity

The function `user_Cp` in "rf_user_Cp.c" computes a user-defined specific heat $\hat{C}_p$ of a non-`CHEMKIN` material. It is called when the following line is included in the Materials Properties section of the input file:

```
CP = VARIABLE_PROP
```

The value of the specific heat is returned by `user_Cp` in the argument `*cp`. Other arguments passed to `user_Cp` are listed in Table 4.1.

| Argument | Description |
|---|---|
| double temperature | Temperature at position $(x, y, z)$. |
| double X_k[] | Vector of mole fractions at position $(x, y, z)$ indexed by the species number. |
| double Ptherm | Thermodynamic pressure. |
| double x, y, z | Coordinates of the current position. |
| MATSTRUCT_PTR matID_ptr | Pointer to the material property structure for the material. |

*Table 4.1. Arguments passed to user-defined property functions* `user_Cp`, `user_cond`, `user_density` *and* `user_visc`.

### 4.1.2. Thermal Conductivity

The function `user_cond` in "rf_user_cond.c" computes a user-defined value of thermal conductivity $\lambda$ for a non-CHEMKIN material. It is called when the following line is included in the Materials Properties section of the input file:

```
THERMAL_CONDUCT = VARIABLE_PROP
```

The value of the thermal conductivity is returned by `user_cond` in the argument `*conductivity`. Other arguments passed to `user_cond` are listed in Table 4.1.

### 4.1.3. Density

The function `user_density` in "rf_user_density.c" computes a user-defined value of density $\rho$ for a non-CHEMKIN material. It is called when the following line is included in the Materials Properties section of the input file:

```
DENSITY = VARIABLE_PROP
```

The value of the density is returned by `user_density` in the argument `*density`. Other arguments passed to `user_density` are listed in Table 4.1.

### 4.1.4. Viscosity

The function `user_visc` in "rf_user_visc.c" computes a user-defined value of the viscosity $\mu$ for a non-CHEMKIN material. It is called when the following line is included in the Materials Properties section of the input file:

```
VISCOSITY = VARIABLE_PROP
```

The value of the viscosity is returned by `user_visc` in the argument `*viscosity`. Other arguments passed to `user_visc` are listed in Table 4.1.

### 4.1.5. Volumetric Source Terms

Variable volumetric source terms for temperatures and mass fractions are specified in the input file as

Q_VOLUME_VAR = *function_name*

and

Y_VOLUME_VAR = *function_name* {SINGLE | MULTIPLE}.

The related user functions included in MPSalsa are `user_source` for temperatures and `SINGLE` mass fraction source terms and `user_source_multi` for `MULTIPLE` mass fraction source terms.

The `user_source` function returns the value of the source term for one equation. Its prototype is

```
double user_source (SNGLVAR_FUNCTION_ARGLIST)
```

where `SNGLVAR_FUNCTION_ARGLIST`, as defined in "rf_salsa.h," is described in Table 4.2. For `SINGLE` source term functions, the boundary condition pointer `bc` is `NULL`.

| Argument | Description |
|---|---|
| double soln[] | Solution vector at position $(x, y, z)$. |
| double x, y, z | Coordinates of position $(x, y, z)$. |
| double t | Time. |
| MATSTRUCT_PTR matID_ptr | Pointer to the material property structure for the material being processed (defined in "rf_matrl_const.h"). |
| int var_num | Equation for which to compute a value (e.g., TEMPERATURE, VELOCITY1, MASS_FRACTION) as defined in "rf_fem_const.h." |
| int sub_var_num | Species for which to compute a value (applicable only when var_num == MASS_FRACTION). |
| int eqn_offset[] | Offset into soln[] for each variable; e.g., the temperature at $(x,y,z)$ is soln[eqn_offset[TEMPERATURE]]. |
| int num_dim | Number of dimensions in the element. |
| BCSTRUCT_PTR bc | Pointer to the boundary condition structure (defined in "rf_bc_const.h") for the current boundary condition being processed. This pointer is NULL if the SNGLVAR_FUNCTION function is called for a calculation not involving a boundary condition. |

*Table 4.2. Arguments included in* `SNGLVAR_FUNCTION_ARGLIST`.

An example of `user_source` is included in Figure 4.1. This function is stored in "rf_user_source_fn.c." To add new user-defined source functions, users should write the functions in either "rf_source_fn.c" or "rf_user_source_fn.c," include prototypes for the new functions in "rf_source_fn_const.h," and add pointer assignments for the new functions to the routine `align_single_q_ptr` in "rf_source_fn.c." Users can look at prototypes and pointer assignments for `user_source` as examples for their own functions.

To reduce the number of function calls needed to compute source terms for mass fraction equations, `user_source_multi` may be used. While `user_source` returns only a single source term value, `user_source_multi` returns a vector of source terms for mass fraction equations. The prototype for `user_source_multi` is

```
void user_source_multi (MULTIVAR_FUNCTION_ARGLIST)
```

where `MULTIVAR_FUNCTION_ARGLIST` is described in Table 4.3.

```
double user_source(SNGLVAR_FUNCTION_ARGLIST)
{
/* Returns the source terms for the coupled linear diffusion equations:
 *
 *       $\nabla^2 T - a = 0$
 *
 *       $\nabla^2 Y_0 + Y_1 - Y_2 = 0$
 *
 *       $\nabla^2 Y_1 - Y_0 e^{-x} = 0$
 *
 *       $\nabla^2 Y_2 - Y_0 - a = 0$
 *
 * where $a = 2$ in 1D, $a = 4$ in 2D, and $a = 6$ in 3D.
 *
 * USAGE: In Material Properties section...
 *      Q_VOLUME_VAR = user_source
 *      Y_VOLUME_VAR = user_source SINGLE
 */
    double return_value;
    double spatial_coeff = 2 * num_dim;

    if (var_num == TEMPERATURE)
        return_value = -spatial_coeff;
    else if (var_num == MASS_FRACTION && sub_var_num <= 2)
        switch (sub_var_num) {
            case 0:
                return_value = soln[eqn_offset[MASS_FRACTION + 1]]
                             - soln[eqn_offset[MASS_FRACTION + 2]];
                break;
            case 1:
                return_value = -soln[eqn_offset[MASS_FRACTION] * exp(-x);
                break;
            case 2:
                return_value = -spatial_coeff -soln[eqn_offset[MASS_FRACTION]];
                break;
        }
    else {
        (void) fprintf(stderr, "ERROR in use of user_source.\n");
        exit(-1);
    }
    return (return_value);
}
```

*Figure 4.1. Example of function* `user_source` *computing volumetric source terms for temperature and mass fraction equations.*

An example of `user_source_multi` is included in Figure 4.2. This function computes the same mass fraction source terms in one function call that function `user_source` in Figure 4.1 would compute in three separate calls.

The function `user_source_multi` is stored in "rf_user_source_fn.c." Users may add their own `MULTIPLE` source functions to either "rf_source_fn.c" or "rf_user_source_fn.c." Prototypes for the new functions should be included in "rf_source_fn_const.h," and pointer assignments must be added to the routine `align_multi_q_ptr` in "rf_source_fn.c." Users can look at prototypes and pointer assignments for `user_source_multi` as examples for their own functions.

```
void user_source_multi(MULTIVAR_FUNCTION_ARGLIST)
{
/*  Returns (in src_vec[]) the source terms for the coupled linear
 *  diffusion equations:
 *
 *       ∇²Y₀ + Y₁ - Y₂ = 0
 *
 *       ∇²Y₁ - Y₀e^(-x) = 0
 *
 *       ∇²Y₂ - Y₀ - a = 0
 *
 *  where a = 2  in 1D,  a = 4  in 2D, and a = 6  in 3D
 *
 * USAGE: In Material Properties section...
 *      Y_VOLUME_VAR = user_source_multi MULTIPLE
 */
   double spatial_coeff = 2 * num_dim;
   int eqnY_offset = eqn_offset[MASS_FRACTION];

   src_vec[0] = soln[eqnY_offset+1] - soln[eqnY_offset+2];
   src_vec[1] = -soln[eqnY_offset] * exp(-x);
   src_vec[2] = -spatial_coeff - soln[eqnY_offset];
}
```

*Figure 4.2. Example of function* `user_source_multi` *computing volumetric source terms for mass fraction equations.*

| Argument | Description |
|---|---|
| double src_vec[] | Returned vector of source term values at $(x, y, z)$, with one value for each mass fraction equation. |
| double soln[] | Solution vector at position $(x, y, z)$. |
| double x, y, z | Coordinates of position $(x, y, z)$. |
| double t | Time. |
| MATSTRUCT_PTR matID_ptr | Pointer to the material property structure (defined in "rf_matrl_const.h") for the material being processed. |
| int eqn_offset[] | Offset into soln[] for each variable; e.g., the temperature at $(x,y,z)$ is soln[eqn_offset[TEMPERATURE]]. |
| int num_dim | Number of dimensions in the element. |

*Table 4.3. Arguments included in* `MULTIVAR_FUNCTION_ARGLIST`.

Analytic Jacobian entries for variable volumetric temperature and mass fraction source terms are specified in the Materials Specifications section of the input file as

`JACOBIAN_SRC_TERMS_VAR` = *function_name*

where *function_name* is a function computing a matrix of derivatives of the source terms with respect to temperature and mass fractions. The user function `user_jac_src` is provided for this purpose. The prototype for `user_jac_src` is

```
void user_jac_src (JAC_SRC_FUNCTION_ARGLIST)
```

where JAC_SRC_FUNCTION_ARGLIST is described in Table 4.4. The derivatives of the source terms are returned in the matrix jac_vec, where jac_vec[*i*][*j*] is the derivative of the source term for the $j^{th}$ equation with respect to the $i^{th}$ variable.

| Argument | Description |
|---|---|
| double *jac_vec[] | Returned matrix of analytic Jacobian terms of source term values with respect to temperature and mass fractions; jac_vec[i][j] is the derivative of the source term for the $j^{th}$ equation with respect to the $i^{th}$ variable. |
| double soln[] | Solution vector at position $(x, y, z)$. |
| double x, y, z | Coordinates of position $(x, y, z)$. |
| double t | Time. |
| MATSTRUCT_PTR matID_ptr | Pointer to the material property structure (defined in "rf_matrl_const.h") for the material being processed. |
| int eqn_offset[] | Offset into soln[] for each variable; e.g., the temperature at $(x,y,z)$ is soln[eqn_offset[TEMPERATURE]]. |
| int num_dim | Number of dimensions in the element. |

*Table 4.4. Arguments included in* JAC_SRC_FUNCTION_ARGLIST.

Figure 4.3 includes an example of user_jac_src that computes the Jacobian entries for the source terms in function user_source in Figure 4.1. This function is stored in "rf_user_jac_src_fn.c." To add new user-defined analytic Jacobian functions for source terms, users should write the functions in either "rf_jac_src_fn.c" or "rf_user_jac_src_fn.c," include prototypes for the new functions in "rf_source_fn_const.h," and add pointer assignments for the new functions to the routine align_jac_src_ptr in "rf_jac_src_fn.c." Users can look at prototypes and pointer assignments for user_jac_src as examples for their own functions.

The following run-time error messages alert users to incorrect implementation of user source term and Jacobian entry functions.

> ERROR: Unknown name for volumetric source function: *function_name*
> ERROR: Unknown name for analytic Jacobian of source vector function: *function_name*

The first message indicates an error with a function specified by Q_VOLUME_VAR or Y_VOLUME_VAR in the input file; the second indicates an error with a function specified by JACOBIAN_SRC_TERMS_VAR. In both cases, a function name is either misspelled in the input file or not added correctly to the pointer assignment routines.

```
void user_jac_src (JAC_SRC_FUNCTION_ARGLIST)
{
/*  Returns (in jac_vec[]) the analytic Jacobian entries of source terms
 *  with respect to (w.r.t.) temperature and mass fractions
 *  for the coupled linear diffusion equations:
 *
 *       $\nabla^2 T - a = 0$
 *
 *       $\nabla^2 Y_0 + Y_1 - Y_2 = 0$
 *
 *       $\nabla^2 Y_1 - Y_0 e^{-x} = 0$
 *
 *       $\nabla^2 Y_2 - Y_0 - a = 0$
 *
 *  where $a = 2$ in 1D, $a = 4$ in 2D, and $a = 6$ in 3D
 *
 * USAGE: In Material Properties section...
 *      JACOBIAN_SRC_TERMS_VAR = user_jac_src
 */
int indxT = eqn_offset[TEMPERATURE], indxY = eqn_offset[MASS_FRACTION];

    /** Derivative of TEMPERATURE source term w.r.t. TEMPERATURE. **/
    jac_vec[indxT][indxT] += 0.0;

    /** Derivatives of MASS_FRACTION src terms w.r.t. TEMPERATURE.**/
    jac_vec[indxT][indxY]   += 0.0;
    jac_vec[indxT][indxY+1] += 0.0;
    jac_vec[indxT][indxY+2] += 1.0;

    /** Derivative of TEMPERATURE source term w.r.t. Y_0. **/
    jac_vec[indxY][indxT] += 1.0;

    /** Derivatives of MASS_FRACTION source terms w.r.t. Y_0. **/
    jac_vec[indxY][indxY]   += 0.0;
    jac_vec[indxY][indxY+1] += -exp(-x);
    jac_vec[indxY][indxY+2] += -1.0;

    /** Derivative of TEMPERATURE source term w.r.t. Y_1. **/
    jac_vec[indxY+1][indxT] += 1.0;

    /** Derivatives of MASS_FRACTION source terms w.r.t. Y_1. **/
    jac_vec[indxY+1][indxY]   += 1.0;
    jac_vec[indxY+1][indxY+1] += 0.0;
    jac_vec[indxY+1][indxY+2] += -1.0;

    /** Derivative of TEMPERATURE source term w.r.t. Y_2. **/
    jac_vec[indxY+2][indxT] += -1.0;

    /** Derivatives of MASS_FRACTION source terms w.r.t. Y_2. **/
    jac_vec[indxY+2][indxY]   += -1.0;
    jac_vec[indxY+2][indxY+1] +=  0.0;
    jac_vec[indxY+2][indxY+2] +=  0.0;
}
```

*Figure 4.3. Example of function* `user_jac_src` *computing analytic Jacobian entries of source terms with respect to temperature and mass fractions for the source function in Figure 4.1.*

## 4.2. Boundary Conditions

User functions may be used for several parts of the Boundary Condition Specifications described in Section 3.7.2. The user function designed to compute boundary condition values is `user_bc_exact`. The prototype for `user_bc_exact` is

```
        double user_bc_exact(SNGLVAR_FUNCTION_ARGLIST)
```

where `SNGLVAR_FUNCTION_ARGLIST` is described in Table 4.2. All arguments of `SNGLVAR_FUNCTION_ARGLIST` are used for boundary condition functions.

An example demonstrating the usage of `user_bc_exact` is given in Figure 4.4. This function is stored in "rf_user_bc_exact_fn.c." To add new user-defined boundary condition functions, users should write the functions in either "rf_bc_exact_fn.c" or "rf_user_bc_exact_fn.c," include prototypes for the new functions in "rf_bc_exact_fn_const.h," and add pointer assignments for the new functions to the routine `align_f_ptr` in "rf_bc_exact_fn.c." Users can look at prototypes and pointer assignments for `user_bc_exact` as examples for their own functions.

Jacobian entries associated with boundary conditions can be specified by the user function `user_jac_bc`. The prototype for `user_jac_bc` is

```
        double user_jac_bc (JAC_BC_FUNCTION_ARGLIST)
```

where `JAC_BC_FUNCTION_ARGLIST` is described in Table 4.5.

Figure 4.5 contains an example of `user_jac_bc` for the boundary conditions specified by `user_bc_exact` in Figure 4.4. This function is stored in "rf_user_jac_bc_fn.c." To add new user-defined functions for the derivatives of boundary condition functions, users should write the functions in either "rf_jac_bc_fn.c" or "rf_user_jac_bc_fn.c," include prototypes for the new functions in "rf_bc_exact_fn_const.h," and add pointer assignments for the new functions to the routine `align_jbc_ptr` in "rf_jac_bc_fn.c." The prototypes and pointer assignments for `user_jac_bc` serve as examples for new user functions for boundary condition derivatives.

The following run-time error messages alert users to incorrect implementation of user-defined boundary condition functions.

> ERROR: Unknown SNGLVAR_FUNCTION: *function_name*
> ERROR: Unknown JAC_BC_FUNCTION: *function_name*

The first message indicates an error in a boundary condition function name; the second indicates an error in the function name for Jacobian entries of a boundary condition. In both cases, a function name was either misspelled in the input file or not added correctly to the appropriate pointer alignment routine.

```
double user_bc_exact(SNGLVAR_FUNCTION_ARGLIST)
{
/* Returns the following Dirichlet boundary conditions for coupled
 * linear diffusion equations:
 *
 *      T = x^2 + y^2 + z^2
 *
 *      Y_0 = ae^x  where  a = 2  in 1D,  a = 4  in 2D, and  a = 6  in 3D
 *
 *      Y_1 = T
 *
 *      Y_2 = Y_0 + Y_1
 *
 * USAGE: In Boundary Conditions section...
 *      BC = T_BC DIRICHLET SS 1 INDEPENDENT user_bc_exact 0
 *      BC = Y_BC DIRICHLET SS 1 INDEPENDENT user_bc_exact 0
 *      SPECIES_LIST = 1
 *      BC = Y_BC DIRICHLET SS 1 DEPENDENT user_bc_exact 0
 *      SPECIES_LIST = 2 3
 */
    double return_value, spatial_coeff;

    if (var_num == TEMPERATURE) {
        return_value = x*x;
        if (num_dim > 1) return_value += y*y;
        if (num_dim > 2) return_value += z*z;
    }
    else if (var_num == MASS_FRACTION && sub_var_num <= 2) {
        switch (sub_var_num) {
            case 0:
                spatial_coeff = 2. * num_dim;
                return_value = spatial_coeff * exp(x);
                break;
            case 1:
                return_value = soln[eqn_offset[TEMPERATURE]];
                break;
            case 2:
                return_value = soln[eqn_offset[MASS_FRACTION]]
                                + soln[eqn_offset[MASS_FRACTION + 1]];
                break;
        }
    }
    else {
        (void) fprintf(stderr, "ERROR in use of user_bc_exact.\n");
        exit(-1);
    }
    return (return_value);
}
```

*Figure 4.4. Example of function* `user_bc_exact` *used as a boundary condition function.*

### 4.2.1. Accessing BC_DATA in User Functions

Each boundary condition in the input file has a `Boundary_Condition` structure (defined in "rf_bc_const.h") associated with it. This structure contains the constant values, pointers to boundary condition functions (such as `user_bc_exact` and those in Appendix A.1), and BC_DATA associated with the boundary condition. Each type of BC_DATA is stored in a one-dimensional array of that type. Integer data, specified by BC_DATA=INT, are stored in the

```
double user_jac_bc(JAC_BC_FUNCTION_ARGLIST)
{
/*  Returns the derivatives of the following Dirichlet boundary
 *  conditions for coupled linear diffusion equations:
 *
 *        $T = x^2 + y^2 + z^2$
 *
 *        $Y_0 = at^r$  where  $a = 2$  in 1D,  $a = 4$  in 2D, and  $a = 6$  in 3D
 *
 *        $Y_1 = T$
 *
 *        $Y_2 = Y_0 + Y_1$
 *
 * USAGE: In Boundary Conditions section...
 *      BC = T_BC DIRICHLET SS 1 INDEPENDENT user_bc_exact 0
 *      BC = Y_BC DIRICHLET SS 1 INDEPENDENT user_bc_exact 0
 *      SPECIES_LIST = 1
 *      BC = Y_BC DIRICHLET SS 1 DEPENDENT user_jac_bc user_bc_exact 0
 *      SPECIES_LIST = 2 3
 */
double return_value = 0.0;

    /*  TEMPERATURE BC does not depend on other variables.
     *  Y_0 BC does not depend on other variables.
     *  Y_1 BC does not depend on other mass fractions.
     *  Y_2 BC does not depend on temperature.
     *  return_value is already set to zero for these entries.
     */

    if (var_num == MASS_FRACTION && sub_var_num <= 2) {
        switch (sub_var_num) {
            case 1:
                if (wrt_var_num == TEMPERATURE)
                    /* Derivative of Y_1 BC w.r.t. TEMPERATURE is 1.0. */
                    return_value = 1.0;
                break;
            case 2:
                if (wrt_var_num == MASS_FRACTION)
                    if (wrt_sub_var_num == 0 || wrt_sub_var_num == 1)
                        /* Derivative of Y_2 BC w.r.t. Y_0 or Y_1 is 1.0. */
                        return_value = 1.0;
                break;
        }
    }
    else if (var_num != TEMPERATURE) {
        (void) fprintf(stderr, "ERROR in use of user_jac_bc.\n");
        exit(-1);
    }
    return (return_value);
}
```

*Figure 4.5. Example of user function* `user_jac_bc` *that computes derivatives of the boundary conditions in* `user_bc_exact` *in Figure 4.4.*

integer array `BC_Data_Int` in the `Boundary_Condition` structure; floating point data, specified by `BC_DATA=FLOAT`, are stored in the double array `BC_Data_Float`; and function pointer data, specified by `BC_DATA=FUNCTION`, are stored in the `BC_Data_User_Fn_Ptr` array. The data are stored in the order they appear in the input file, starting from array index 0 in each array.

| Argument | Description |
|---|---|
| double soln[] | Solution vector at position (*x, y, z*). |
| double x, y, z | Coordinates of position (*x, y, z*). |
| double t | Time. |
| MATSTRUCT_PTR matID_ptr | Pointer to the material property structure (defined in "rf_matrl_const.h") for the material being processed. |
| int var_num | Dependent variable of the partial derivative (e.g., TEMPERATURE, VELOCITY1, MASS_FRACTION) as defined in "rf_fem_const.h." |
| int sub_var_num | Species number for the dependent variable of the partial derivative (applicable only when var_num == MASS_FRACTION). |
| int wrt_var_num | Independent variable of the partial derivative to be taken (e.g., TEMPERATURE, VELOCITY1, MASS_FRACTION) as defined in "rf_fem_const.h" |
| int wrt_sub_var_num | Species number for the independent variable of the partial derivative (applicable only when wrt_var_num == MASS_FRACTION). |
| int eqn_offset[] | Offset into soln[] for each variable; e.g., the temperature at (*x,y,z*) is soln[eqn_offset[TEMPERATURE]]. |
| int num_dim | Number of dimensions in the element. |
| BCSTRUCT_PTR bc | Pointer to the boundary condition structure (defined in "rf_bc_const.h") corresponding to the current boundary condition being processed. |

*Table 4.5. Arguments included in* JAC_BC_FUNCTION_ARGLIST.

The argument bc in SNGLVAR_FUNCTION_ARGLIST and JAC_BC_FUNCTION_ARGLIST is a pointer to the Boundary_Condition structure associated with the boundary condition. BC_DATA can be accessed by following this pointer. For example, the first BC_DATA=INT value entered in the input file would be accessed in boundary condition functions by bc->BC_Data_Int[0]. An example boundary condition function using BC_DATA is included in Figure 4.6. In this example, the rotation rate and center of rotation of a two-dimensional disk are given by BC_DATA=FLOAT values in the input file.

Functions listed in BC_DATA=FUNCTION lines must also be boundary condition functions as described in Section 4.2. They must have the same prototypes as user_bc_exact and be called with the SNGLVAR_FUNCTION_ARGLIST argument list in Table 4.2. As with all user boundary condition functions, they must be included in the pointer assignment routine align_f_ptr and compiled into MPSalsa. The syntax for calling, say, the second BC_DATA=FUNCTION listed for a boundary condition is shown below:

```
val = bc->BC_Data_User_Fn_Ptr[1](soln, x, y, z, t,
                   matID_ptr, var_num, sub_var_num, eqn_offset,
                   num_dim, bc);
```

```
double f_xy_spin_disk (SNGLVAR_FUNCTION_ARGLIST)
{
/* Function to return value of the x,y velocity on a rotating disk.
 * This function takes 3 arguments:
 *      BC_Data_Float[0] = rotation rate in rpm, counter clockwise
 *      BC_Data_Float[1] = x_0
 *      BC_Data_Float[2] = y_0
 *
 * Usage: e.g. Disk spinning at 50rpm around x=0, y=0
 *      U_BC DIRICHLET NS 1 INDEPENDENT f_xy_spin_disk 1
 *      BC_DATA = 50.0 0.0 0.0
 *      V_BC DIRICHLET NS 1 INDEPENDENT f_xy_spin_disk 1
 *      BC_DATA = 50.0 0.0 0.0
 */
double omega = 0.0, x_0 = 0.0, y_0 = 0.0; /* default values */
double x_offset, y_offset, result;

    /* Use BC_DATA values if any are specified in the input file. */

    if (bc->BC_Data_Float != NULL) {
        /* Conversion from rpm to radians/sec done once in bc_input_pre_process */
        /*   omega = (bc->BC_Data_Float[0] * 2.0 * pi)/60.0;   */
        omega =  bc->BC_Data_Float[0];
        x_0   =  bc->BC_Data_Float[1];
        y_0   =  bc->BC_Data_Float[2];
    }

    x_offset = (x - x_0);
    y_offset = (y - y_0);

    if      (var_num == VELOCITY1) result =  (-omega * y_offset);
    else if (var_num == VELOCITY2) result =  ( omega * x_offset);
    else if (var_num == TANGENT_VELOCITY1)
        /* Assumes t1 = [0.8, 0.6, 0.0]  */
        result = 0.8 * (-omega * y_offset) + 0.6 * ( omega * x_offset);
    else if (var_num == TANGENT_VELOCITY2)
        /* Assumes t2 = [-0.6, 0.8, 0.0]  */
        result = -0.6 * (-omega * y_offset) + 0.8 * ( omega * x_offset);

    return (result);
}
```

*Figure 4.6. Example demonstrating the use of* BC_DATA *in boundary condition functions.*

## 4.3. Generalized Surfaces

User-defined outward normal and tangent vectors may be specified through the use of generalized surfaces as described in Section 3.7.1. The functions `user_normal`, `user_tangent1`, and `user_tangent2` are provided for this purpose. They return the appropriate surface vector as a function of position on the surface. The prototypes for these functions are

```
void user_normal (SURF_VECTOR_FUNCTION_ARGLIST)
void user_tangent1 (SURF_VECTOR_FUNCTION_ARGLIST)
void user_tangent2 (SURF_VECTOR_FUNCTION_ARGLIST)
```

where SURF_VECTOR_FUNCTION_ARGLIST is defined in "rf_bc_const.h" and described in Table 4.6.

| Argument | Description |
|---|---|
| double surf_vec[] | Returned vector containing the *x*-, *y*-, and *z*-components of a surface vector. |
| double x, y, z | Coordinates of position (*x*, *y*, *z*). |

*Table 4.6. Arguments included in* SURF_VECTOR_FUNCTION_ARGLIST.

Examples of the generalized surface functions are given in Figure 4.7. The functions are stored in "rf_user_tangent_fn.c." To add new user-defined functions for describing generalized surfaces, users should write the functions in either "rf_tangent_fn.c" or "rf_user_tangent_fn.c," include prototypes for the new functions in "rf_tangent_fn.c," and add pointer assignments for the new functions to the routine `align_surf_vector_ptr` in "rf_tangent_fn.c." The prototypes and pointer assignments for `user_normal` serve as examples for newly written user functions for outward normal and tangent vectors.

The following run-time error message alerts users to incorrect implementation of user-defined normal and tangent functions:

> ERROR - unknown surface vector function: *function_name*

A function name was either misspelled in the input file or not added correctly to the `align_surf_vector_ptr` routine.

## 4.4. Initial Condition/Guess

Initial guesses may be specified through the `user_init_cond` function. The prototype for `user_init_cond` is

```
double user_init_cond (SNGLVAR_FUNCTION_ARGLIST)
```

where SNGLVAR_FUNCTION_ARGLIST is described in Table 4.2. The arguments `matID_ptr` and bc in SNGLVAR_FUNCTION_ARGLIST are NULL when a function is used as an initial condition function. The function `user_init_cond` is in file "rf_user_init_cond_fn.c." New initial condition functions should be added to this file or to "rf_bc_exact_fn.c." Prototypes for new functions should be added to "rf_bc_exact_fn_const.h," and function pointers must be added to `align_f_ptr` in "rf_bc_exact_fn.c."

```
void user_normal(SURF_VECTOR_FUNCTION_ARGLIST)
{
/*
 * Outward normal vector (along circle of radius one) of cylinder aligned
 * in z-direction.
 *
 * USAGE: in Generalized Surfaces section ...
 *      NORMAL = user_normal
 */
    surf_vec[0] = x;
    surf_vec[1] = y;
    surf_vec[2] = 0.0;
}

void user_tangent1(SURF_VECTOR_FUNCTION_ARGLIST)
{
/*
 * Tangent vector (along circle of radius one) of cylinder aligned
 * in z-direction.
 *
 * USAGE: in Generalized Surfaces section ...
 *      TANGENT = user_tangent1
 */
    surf_vec[0] = -y;
    surf_vec[1] =  x;
    surf_vec[2] =  0.0;
}

void user_tangent2(SURF_VECTOR_FUNCTION_ARGLIST)
{
/*
 * Tangent vector (along height of cylinder) of cylinder aligned in z-direction.
 *
 * USAGE: in Generalized Surfaces section ...
 *      TANGENT = user_tangent2
 */
    surf_vec[0] = 0.0;
    surf_vec[1] = 0.0;
    surf_vec[2] = 1.0;
}
```

*Figure 4.7. Example of functions* `user_normal`, `user_tangent1`, *and* `user_tangent2` *for generalized surfaces.*

## 4.5. Exact Solutions

For problems having analytic solutions, MPSalsa can compare the computed solution with the analytic solution. The user function `user_bc_exact` in "rf_user_bc_exact_fn.c" may be used to specify the exact solution function. The prototype for `user_bc_exact` is

```
double user_bc_exact (SNGLVAR_FUNCTION_ARGLIST)
```

where `SNGLVAR_FUNCTION_ARGLIST` is described in Table 4.2. Since exact solutions depend only on position and time, the arguments `matID_ptr`, `bc`, and `eqn_offset[]` in `SNGLVAR_FUNCTION_ARGLIST` are NULL when they are arguments to an exact solution function. An example of `user_bc_exact` used as an exact solution function is given in Figure

4.8. The procedures for adding new exact solution functions to MPSalsa are the same as those described in Section 4.2 for adding new boundary condition functions.

```
double user_bc_exact(SNGLVAR_FUNCTION_ARGLIST)
{
/* Returns the exact solution values for the coupled linear diffusion equations:
 *
 *      $T = x^2 + y^2 + z^2$
 *
 *      $Y_0 = ae^x$  where $a = 2$ in 1D, $a = 4$ in 2D, and $a = 6$ in 3D
 *
 *      $Y_1 = x^2 + y^2 + z^2$
 *
 *      $Y_2 = ae^x + x^2 + y^2 + z^2$
 *
 * USAGE: in Output Specification section...
 *      Test Exact Solution Flag = 1
 *      Name of Exact Solution Function = user_bc_exact
 */
    double return_value, spatial_coeff, sum;

    spatial_coeff = 2 * num_dim;
    sum = x*x;
    if (num_dim > 1) sum += y*y;
    if (num_dim > 2) sum += z*z;

    if (var_num == TEMPERATURE) {
        return_value = sum;
    }
    else if (var_num == MASS_FRACTION && sub_var_num <= 2) {
        switch (sub_var_num) {
            case 0:
                return_value = spatial_coeff * exp(x);
                break;
            case 1:
                return_value = sum;
                break;
            case 2:
                return_value = spatial_coeff * exp(x) + sum;
                break;
        }
    }
    else {
        (void) fprintf(stderr, "ERROR in use of user_bc_exact.\n");
        exit(-1);
    }
    return (return_value);
}
```

*Figure 4.8. Example of function* `user_bc_exact` *used as an exact solution function.*

## 4.6. Output

Functions can be written to compute specific output from the solution. At the initial conditions, after every time step, and after calculating a steady-state solution, the function `user_out` in the file "rf_user_out.c" is called. The default function `user_out` computes the maximum, minimum, and average value of each variable as well as the position of the maximum and minimum. (Little investment has been made in providing output options for MPSalsa since commercial visualization packages that read in the FE mesh and solutions from the ExodusII database have satisfied most of our post-processing needs.)

Writing additional output routines should be done using the `user_out` function, either by replacing it with an alternate function or by calling another function from within it. The second option was chosen for implementing routines such as `time_history_points` (See Section A.3.1).

The `status` integer flag passed to `user_out` contains information on whether the solution is an initial guess, an intermediate time step, a failed time step, or a final solution. The values of the flag are shown in Figure 4.9.

```
*   Values for status variable:
*  ------------------------------
*     <0   = Some sort of error condition has occurred.
*      0   = Initial conditions
*      1   = Final conditions, i.e., a successful run has completed
*      2   = A successful intermediate time step has occurred.
```

*Figure 4.9. Values of the* `status` *flag as passed to* `user_out`.

To write new output functions, it is best to modify the default `user_out` or one of the output functions listed in Appendix A. Many quantities that might be useful in output routines -- such as the values of physical properties at the nodes and useful bookkeeping arrays -- are unfortunately not readily available to the output routines. These quantities are stored in memory only during the matrix-fill section of the calculation; after the matrix-fill, their memory is freed to provide as much memory as possible for the matrix-solve.

## 4.7. Continuation

The function `user_continuation` in the file "rf_user_continuation.c" is where the continuation parameter is defined. The continuation parameter can be equated to any boundary condition, physical property, or a combination of these quantities. The function takes as input the pointer to the continuation parameter, and updates the appropriate physical quantity or boundary condition. For instance, if the user would like to continue with respect to the viscosity of the first material, stored globally as `MatID_Prop->viscosity`, `user_continuation` would simply contain the appropriate assignment statement as shown in Figure 4.10.

Similarly, if the user would like to continue with respect to the value of the sixth boundary condition listed in the file, stored globally as `BC_Types[5].BC_Fn_Value`, `user_continuation` would contain just the following assignment statement:

```
function void user_continuation(double *con_par);
/* con_par is a pointer to the continuation parameter */
/* *con_par is the value of the continuation parameter */
{
    MatID_Prop->viscosity = *con_par;
}
```

*Figure 4.10. Example of the function* `user_continuation` *for assigning the continuation parameter to a physical quantity (in this case the fluid viscosity).*

```
BC_Types[5].BC_Fn_Value = *con_par;
```

(Since "C" numbering begins with zero, the sixth boundary condition in the input file is stored in array entry five.)

Another common continuation parameter with boundary conditions is an entry in the `BC_DATA` statement. To continue with respect to the third constant ("C" array entry 2) of the `BC_DATA FLOAT` array of the twenty-third boundary condition ("C" array entry 22), the assignment would be

```
BC_Types[22].BC_Data_Float[2] = *con_par;
```

All parts of the boundary condition structure, not only the `BC_Fn_Value` and `BC_Data_Float[]` examples shown here, can be referenced for use in continuation. The entire structure is listed in the file "rf_bc_const.h." Similarly, the entire materials structure of physical properties can be referenced in the same way the viscosity was above. The structure is defined in the file "rf_matrl_const.h."

The continuation parameter can represent other quantities by more complicated assignment statements. For instance, to continue with respect to the Reynolds number, where the inlet velocity is entered as the fourth BC and the characteristic length is 2.0, the assignment statement would be

```
BC_Types[3].BC_Fn_Value = *con_par * MatID_Prop->viscosity
                              / (2.0 * MatID_Prop->density);
```

In this example, the inlet velocity is manipulated at constant viscosity and density so that the continuation parameter equals the Reynolds number, and other dimensionless numbers stay constant.

## 4.8. Function Data

User data specified in the Function Data section of the input file (see Section 3.11) may be accessed by any of the above user functions. The user function must first locate its particular function data. In the simplest case, the location is found by calling the function `fn_data_location`:

FNDATA_PTR fn_data_location (char yo[], int data_required)

where yo[] is a character string containing the function name associated with the data in the input file, and data_required indicates whether the function data is mandatory or optional. If data_required is TRUE and no function data was included in the input file, MPSalsa will quit with an error condition. When data_required is FALSE, either default values for the data should be supplied or the user function should return immediately without an error.

The function fn_data_location returns a pointer to a Function_Data structure (defined in "rf_fn_data_const.h"). Within the Function_Data structure, Fn_Data_Int, Fn_Data_Float, and Fn_Data_String are arrays of INT, STRING, and FLOAT function data, respectively, from the input file. The numbers of entries in each array are given by Num_Fn_Data_Int, Num_Fn_Data_String and Num_Fn_Data_Float. The arrays are used in a manner analogous to the BC_DATA arrays for boundary conditions (see Section 4.2.1). Data values are stored in the order they are read from the input file, starting from index 0 in the arrays. For example, the fifth string entered as function data would be addressed by current_fn->Fn_Data_String[4]. An example of a boundary condition function that uses optional function data is given in Figure 4.11.

```
double user_bc_exact(SNGLVAR_FUNCTION_ARGLIST)
{
/*
 * Function that returns  $(x-x_0)^2+(y-y_0)^2$  where $x_0$ and $y_0$ may be
 * specified by the user in the function data section of the input file.
 *
 * USAGE: in Function Data Specification section ...
 *      Function Name = user_bc_exact 1
 *      FN_DATA = FLOAT 3.0 2.0
 */
FNDATA_PTR current_fn;

    /* Get the pointer to the function data for this function.  */
    /* This function is optional; if no function data is found, */
    /* $x_0$ and $y_0$ are zero. */

    current_fn = fn_data_location("user_bc_exact", FALSE);

    if (current_fn != NULL)
        if (current_fn->Num_Fn_Data_Float > 0)
            x = (x - current_fn->Fn_Data_Float[0]);
        if (current_fn->Num_Fn_Data_Float > 1)
            y = (y - current_fn->Fn_Data_Float[1]);
    }
    return (x * x + y * y);
}
```

*Figure 4.11. Example usage of function data within a user function.*

A table supplied by the FN_DATA=TABLE mechanism is stored in the Function_Data structure as Fn_Data_Table, a two-dimensional array of double precision numbers. Each row of the table in the input file is stored as a row of the array; that is, the $j^{th}$ entry on the $i^{th}$ row of the input table is stored in `Fn_Data_Table`[*i*][*j*]. The numbers of rows and columns in the table are stored in `Fn_Data_Table_Dim[0]` and `Fn_Data_Table_Dim[1]`, respectively. The function `lookup_table_1` in "rf_fn_data.c" provides a good example of the usage of function data tables (see Appendix A.2).

User functions that operate on several sets of function data are often useful. The function `time_history_line`, for example, prints the solution along a line that is described by a function data table. To print time histories along several lines, a function data entry is included in the input file for each line. Such user functions must loop over all the function data and operate on each instance of their function data. The function `fn_data_next_location` is provided to allow processing of two or more sets of function data by a single function. The prototype for `fn_data_next_location` is

```
FNDATA_PTR fn_data_next_location(char yo[],
              int data_required, int start_ifd, int *found_ifd)
```

where `yo[]` is the function name specified in the input file, `data_required` indicates whether the function data are required or optional, `start_ifd` is the first function data entry to be checked for a match with `yo[]`, and the index of the function data entry matching `yo[]` is returned in `found_ifd`. The value of `found_ifd+1` should be used as `start_ifd` in subsequent searches for more function data for `yo[]`. A pointer to the function data indexed by `found_ifd` is returned by `fn_data_next_location`. An example demonstrating the usage of `fn_data_next_location` in a loop over function data is given in Figure 4.12.

```
void function_name( )
{
/*
 * USAGE: in Function Data Specification section ...
 *      Function Name = function_name 1
 *      FN_DATA = STRING data set one
 *      Function Name = function_name 1
 *      FN_DATA = STRING data set two
 */
char yo[] = "function_name";
FNDATA_PTR current_fn = NULL;
int ifd = -1;
extern int Num_Fn_Data; /* Number of function data entries in the input file */

    while (ifd < Num_Fn_Data) {

        /* Get the pointer to the function data for this function */
        current_fn = fn_data_next_location(yo, FALSE, ifd+1, &ifd);

        if (current_fn == NULL) {
            printf("No additional Function Data found for %s\n", yo);
            break;
        }
        else {
            /*
             * Process the data pointed to by current_fn.
             */
        }
    }
}
```

*Figure 4.12. Example usage of* `fn_data_next_location` *to process more than one set of function data within a function.*

# 5. Solution Strategies

## 5.1. Getting to a Steady State

Sometimes a steady-state solution to a non-linear problem is desired but MPSalsa will not converge to it for a given input file and a simple initial guess. The following is a list of some input file options and techniques that can help. Some of the options are discussed in more detail later in this chapter.

(1) Increase the maximum number of Newton iterations. (See Section 3.3.1.)

(2) Choose a more robust preconditioner, such as `no_overlap_bilu` or `real_overlap_ilu`. If the program runs out of memory, use a larger number of processors. (See Section 3.3.2.)

(3) Increase the number of Krylov subspace vectors for GMRES. If the program runs out of memory, use a larger number of processors. For problems of a few hundred thousand unknowns, a Krylov subspace size over 100 is desirable. (See Section 3.3.2.)

(4) Switch the `Enable backtracking for residual reduction` flag from `on` to `off`, or from `off` to `on`. We have seen examples where the problem converges only with backtracking `on`, and we have seen cases that converge only with backtracking `off`. (See Section 3.3.1.)

If none of the easy solutions above works, the following options may.

(5) Use pseudo time-stepping as the `Solution Type` to relax the system. If the initial time step is small and `Time Step Control` is `on`, pseudo time stepping increases the time step for any step that converges, regardless of integration error. After 5-20 successful time steps have been taken, one can often restart from the last time step and converge to the steady-state. (See Section 3.2.)

(6) Use the restart capability to step to the solution by first solving the problem at simpler conditions, such as at a reduced density or thermodynamic pressure, an elevated viscosity, or with reactions turned off using the `Species equation source terms` and `Energy equation source terms` flags. Then, use this intermediate solution as an initial guess for the desired solution. (See Section 3.8.)

(7) Use continuation to automatically step through a series of steady states as a single parameter is incremented until reaching the desired conditions. (See Section 5.4.)

(8) Do mesh sequencing to first solve the problem on a coarse mesh, and work toward a fine mesh. Convergence is often better on coarse meshes because the preconditioners span more of the domain. (See Section 5.3.)

(9) Write an initial guess function with an educated guess of what the solution will look like as a function of *x*, *y*, and *z*. (See Section 3.8 and Section 4.4.)

## 5.2. Picking a Linear Solver and Preconditioner

The choices for the linear solver, the preconditioner, and the scaling method are listed in Table 3.3, Table 3.5, and Table 3.6, respectively, and lead to hundreds of possible combinations. In Table 5.1 below, we list the three combinations that we use most often. The most common combination is #1, which does well for getting to a steady-state (i.e., for `steady`, `pseudo`, or `continuation` solution types as listed in Section 3.2). With the GMRES method, the Krylov subspace dimension can be increased to be as big as will fit on the machine without running out of memory (or causing excess swapping on some machines), up to a value of a few hundred. The total number of linear solver iterations should usually be two or three times the Krylov subspace size, since GMRES tends to make little progress after restarting three times.

If a steady state is desired but the job runs out of memory at low values of the Krylov subspace, there are two options: (1) use a larger number of processors, and (2) switch to a different solver such as the tfqmr solver (combination #2).

For `transient` runs where speed is more important than robustness, the scheme #3 is often used. This scheme uses only about half the memory of scheme #1 and the calculation of the scaling matrix is much quicker than an ILU-type preconditioner.

| Scheme, in decreasing order of robustness and memory use | Linear Solver | Preconditioner | Scaling | Krylov subspace |
|---|---|---|---|---|
| 1. Robust; good for Steady-State | gmres | no_overlap_ilu | row_sum | large (>100) |
| 2. Robust; uses less Memory | tfqmr | no_overlap_ilu | row_sum | |
| 3. Fast; Good for Transient | gmres | none | block_Jacobi | moderate |

*Table 5.1. Three common linear solution schemes.*

## 5.3. Mesh Sequencing

Mesh sequencing is a strategy for more easily obtaining steady-state solutions on fine meshes. In mesh sequencing, a solution is first computed on a coarse mesh. This solution is interpolated to a finer mesh and used as the initial guess for the solution on the fine mesh.

Sequences of successively finer meshes can be used until a solution with the desired resolution is obtained.

Using MerlinII [15] to interpolate the solution from coarse meshes to fine ones, we have run some experiments with mesh sequencing in MPSalsa. In Table 5.2, we show results for the Lid-Driven Cavity problem (see Appendix C.2) with an upper-wall velocity of $u = 1500$. Steady-state solutions were obtained with an initial guess of zero for all unknowns and with initial guesses interpolated from coarser meshes. The linear solver was GMRES with an ILU preconditioner. The number of Newton iterations and the solution times on the Intel Paragon are compared.

| Mesh Size | Number of Processors | Initial Guess | Number of Newton Iterations | Execution Time (seconds) | MerlinII's Execution Time (seconds) |
|---|---|---|---|---|---|
| 16x16 | 1 | 0.0 | 13 | 59.2 | |
| 32x32 | 4 | 0.0 | 10 | 73.1 | |
| | | Sol'n from 16x16 | 6 | 46.3 | 1.0 |
| 64x64 | 16 | 0.0 | 11 | 220.0 | |
| | | Sol'n from 16x16 | 8 | 165.5 | 2.7 |
| | | Sol'n from 32x32 | 6 | 122.5 | 5.2 |
| 128x128 | 64 | 0.0 | 39 | 1406.1 | |
| | | Sol'n from 16x16 | 29 | 1025.3 | 9.1 |
| | | Sol'n from 32x32 | 23 | 820.3 | 17.6 |
| | | Sol'n from 64x64 | 17 | 610.8 | 52.4 |

*Table 5.2. Performance of the non-linear solver for the Lid-Driven Cavity example using initial guesses of zero and initial guesses obtained from coarse-mesh solutions.*

MerlinII is included in the SEACAS distribution of utilities for ExodusII. If the SEACAS utilities are installed in directory $ACCESS, the path $ACCESS/etc must be included in the user's path. The command line for MerlinII to interpolate the solution from a coarse mesh to a fine mesh is shown below:

```
> merlin2 -input merlin.inp -output merlin.out -plot coarse_soln.exoII -mesh
  fine_mesh.exoII -interpolate merlin.exoII
```

where "coarse_soln.exoII" is the ExodusII file containing the coarse-mesh solution, "fine_mesh.exoII" is the ExodusII file containing the fine mesh, "merlin.exoII" is the resulting ExodusII file containing the fine-mesh solution interpolated from the coarse-mesh solution, "merlin.out" is a text file containing error messages, if any, and "merlin.inp" is an input file containing processing instructions for MerlinII. The MerlinII input file for the Lid-Driven Cavity example above is shown in Figure 5.1; see [15] for more details.

```
$ INPUT FILE FOR THE LID-DRIVEN CAVITY EXAMPLE
$ Declare that the files to interpolate both from and to are EXODUS files.
MESH-A, EXODUS
MESH-B, EXODUS
$ List the variables to be interpolated.
VARIABLES
VX
VY
Pres
END
$ List the time planes to be interpolated.
TIMEPLANE
ALL
END
$ Perform the interpolation and quit.
EXECUTE
STOP
```

*Figure 5.1. MerlinII input file for mesh sequencing in the Lid-Driven Cavity example.*

## 5.4. Continuation

Continuation methods are used to solve for a series of steady-state solutions as a function of a parameter. These methods are commonly used for analysis to study trends in performance or behavior, as we have studied the effect of the disk spin rate on the CVD reactor performance in Section D.3. Continuation can also be an efficient way of reaching a steady-state solution at conditions where a trivial initial guess is not close enough for Newton's method to converge. For instance, a flow problem can be solved easily at low density, and then the density can be incremented over several steps until reaching the desired conditions.

To implement continuation, the user must edit the function `user_continuation` in the file "rf_user_continuation.c" to associate the continuation parameter with a specific boundary condition or a physical, transport, or kinetic property. This can usually be done by editing only one line of code. For details, see Section 4.7.

Users control the continuation routine through the Solution Specifications section of the input file. An example of this section configured for a continuation run is shown in Figure 5.2. The seven lines in this section specify that (1) we are solving a continuation problem; (2) first-order (a.k.a. Euler-Newton) continuation is to be used; (3) a constant step size is to be used as long as a steady-state solution is reached within the maximum number of Newton iterations; (4) the first solution is for a parameter value of 100.0; (5) the first parameter step is of size 100.0;

(6,7) the run will stop when either 20 continuation steps have been taken or when the parameter value exceeds 1300.0.

```
-------------------------------------------------------------
Solution Specifications
-------------------------------------------------------------
Solution Type                      = continuation
Order of integration/continuation = 1
Step Control                       = off
Initial Parameter Value            = 100.0
Initial Step Size                  = 100.0
Maximum Number of Steps            = 20
Maximum Time or Parameter Value    = 1300.0
```

*Figure 5.2. Sample Solution Specifications section for a continuation run.*

The `Order of integration/continuation` flag can have values of 0, 1, or 2. A value of zero indicates zeroth order continuation, where the solution at step *n* is used as an initial guess for solution *n*+1 at the next parameter value. This type of continuation is just an automation of doing a series of steady-state calculations where, for each calculation, the parameter is changed in the input file between each run and the initial solution value is taken from the previous solution.

First-order continuation (when this flag equals one) requires one additional matrix solve to calculate the derivative of the solution with respect to the parameter at step *n*, and uses this tangent to predict an initial guess at the parameter value for step *n*+1. The resulting improvement in the initial guess using first-order continuation usually saves at least one Newton iteration in converging to the solution at step *n*+1, which makes up for the additional cost of the tangent calculation.

A value of two for this flag indicates pseudo arc-length continuation, a capability that is not currently implemented. This method is a powerful tool in bifurcation analysis as it can track solutions around turning points in the solution branch. In pseudo arc-length continuation, the distance along the solution branch (not the change in the parameter) is chosen, so the parameter value is free to increase or decrease. With our block matrix storage format, we have decided not to implement pseudo arc-length continuation by augmenting the system of equations by one, as is commonly done [38], but to use the method described in the Ph.D. dissertation of Shadid [41]. In this method, the continuation step takes two matrix solves to form the initial guess for step *n*+1, although the same preconditioner can be used for both solves.

The other input file choice that deserves additional mention is the `Step Control` flag. When `Time Step Control` is `off`, the step size is held constant for successful steps (where convergence of the nonlinear solver is reached within the maximum number of allowed Newton iterations) and cut in half when a step is unsuccessful. When `Time Step Control` is on, the step size is increased after each successful step. The increase in step size is larger when the ratio of the number of Newton iterations needed for convergence to the total number of Newton iterations allowed is small. Failed steps cut the step size in half.

## 6. Future Development

The following is a list of development work for the MPSalsa code that is already planned or underway.

- Multicomponent Diffusion: A full multicomponent diffusion option will be added, which will be more accurate than the current mixture-average model, yet much more costly to compute.
- Cylindrical coordinates: For 2D meshes, the capability to solve for axisymmetric solutions will be added, with the option of two or three components of the velocity for problems with fluid flow.
- Multi-Physics: This work will add the ability to solve for different physics, and different numbers of unknowns, in distinct "realms" of the computational domain. For instance, heat transfer can be modeled in the solid walls of a reactor together with the reacting gas flows on the inside.
- Turbulence: Implementation of a $k-\varepsilon$ model for time-averaged turbulence is underway, and an LES (Large Eddy Simulation) model for transient turbulence will follow.
- Adaptive Mesh Refinement and Dynamic Load Balancing: The ability to automatically refine a mesh to reduce a measure of the discretization error below a given tolerance will be added. As elements are created and destroyed nonuniformly, the work load will be redistributed over the processors.
- Stability Analysis: A pseudo arc-length continuation routine will be added to track steady-state solution branches, even if they lose stability through a turning point. To check the stability of steady solutions, the ability to calculate eigenvalues of the Jacobian matrix will be added through ARPACK [47], which we will access through the Aztec library.
- Radiation: The ability to include the radiant energy exchange due to enclosure radiation using the methods in COYOTE II [16] is mostly implemented in MPSalsa. Work is also underway to implement a participating media radiation model.
- Porous Media: The ability to model multiphase flow in porous media has been implemented in a previous version of MPSalsa [32], and will be integrated into the current version in the future. The Brinkman equation, which just requires the addition of drag terms to the Navier-Stokes equations, will also be included.
- Plasma Physics: The ability to model dense, partially ionized plasma/gas mixtures using self-consistent charged species transport models will be added.

## Appendix A. Included Functions

## A.1. Boundary Conditions

### A.1.1. Surface Chemistry Boundary Conditions

Effects due to surface reactions are included through the use of surface chemistry boundary conditions. The function `surface_chemkin_bc` computes the temperature and mass fraction NEUMANN boundary conditions, and Stefan flow `DIRICHLET` velocity boundary conditions below:

$$\mathbf{n} \cdot \mathbf{q}_c = \sum_{k=1}^{N} \dot{s}_k W_k h_k, \tag{A.1}$$

$$\mathbf{n} \cdot \mathbf{j}_k = -\dot{s}_k W_k - (\mathbf{n} \cdot \rho Y_k \mathbf{u}) \text{ , and} \tag{A.2}$$

$$\mathbf{n} \cdot \mathbf{u} = -\frac{1}{\rho} \sum_{k=1}^{N_g} \dot{s}_k W_k, \tag{A.3}$$

where $\dot{s}_k = \dot{s}_k(P, T, \mathbf{Y}, \mathbf{Z})$ is the production rate of gas- or surface-phase species $k$ due to surface reaction, $\mathbf{Z}$ is the vector of surface site fractions, $W_k$ is the molecular weight of species $k$, $h_k$ is the enthalpy of species $k$, $N_g$ is the number of gas-phase species, and $N$ is the total number of gas-, surface-, and bulk-phase species (see [5, 42] for more details of these surface reaction boundary conditions). Examples using the `surface_chemkin_bc` function for (A.1) - (A.3) are included in Figure A.1. The Stefan velocity boundary condition (A.3) may be implemented as either a `VEL_NORM_BC` or as a `U_BC`, `V_BC`, or `W_BC` when the normal vector is parallel to the *x*-, *y*-, or *z*-axis, respectively. In the latter case, the sign of the normal vector will be taken into account automatically.

The initial surface site fractions and bulk species mass fractions may be specified in the input file by including `SURF_SPECIES_LIST` and `BC_DATA` lines with the `surface_chemkin_bc` mass fraction boundary condition. The format for these lines follows:

`SURF_SPECIES_LIST = {ALL` | *list of species numbers* | *list of species names*}
`BC_DATA = FLOAT` *list of surface site fractions or mass fractions*

The arguments of `SURF_SPECIES_LIST` are analogous to those of the `SPECIES_LIST` described in Section 3.7.2.1, with the exception that the numbers or names must correspond to surface or bulk species. These two lines together count as one data line in the *num_data_lines*

```
# Temperature BC of equation (A.1).
      BC = T_BC NEUMANN SS 4 DEPENDENT surface_chemkin_bc 0
# Mass fraction BC of equation (A.2).
      BC = Y_BC DIRICHLET SS 4 DEPENDENT surface_chemkin_bc 2
            SPECIES_LIST = ALL
            SURF_SPECIES_LIST = GaMe(S) Ga(S) GaH(S) AsH(S) AsMe(S) As(S)
            BC_DATA = 1.0e-6 0.5 1.0e-6 1.0e-6 1.0e-6 0.5
            SURF_SPECIES_LIST = Ga-GaAs(D) As-GaAs(D)
            BC_DATA = 1.0 1.0
# Tangential velocity BC with value 0.0.
      BC = U_BC DIRICHLET SS 4 INDEPENDENT 0. 0
      BC = V_BC DIRICHLET SS 4 INDEPENDENT 0. 0
# Normal velocity BC of equation (A.3) (Stefan flow).
      BC = Z_BC DIRICHLET SS 4 DEPENDENT surface_chemkin_bc 0
```

*Figure A.1. Example usage of surface_chemkin_bc for surface reaction boundary conditions on temperature, mass fractions and velocity (where the normal to side set 4 is parallel to the z-axis).*

argument of the BC line (see Section 3.7.2). The example in Figure A.1 uses SURF_SPECIES_LIST to initialize both surface site fractions and bulk mass fractions.

### A.1.2. Danckwerts' Boundary Conditions

Danckwerts' boundary condition can be applied using the included functions f_Danckwerts and f_Danckwerts_X0. Danckwerts' boundary condition is used as an inlet boundary condition when the user wants to specify the total flux of each species into the system, rather than the mole or mass fraction of species at the edge of the domain. This is particularly important in low pressure reacting systems, where the diffusive component of the inlet flux of a species $i$ is significant compared to the convective contribution:

$$\mathbf{j}^i_{total} = \mathbf{j}^i_{diffusive} + \mathbf{j}^i_{convective} \tag{A.4}$$

This boundary condition is also important for matching experimental results, where it is generally the total flux of a species $i$ that is known, not the mole fractions at the edge of the computational domain.

It is assumed that the user knows the total flux of each species into the system in terms of the upstream velocity $\mathbf{u}_0$, the normal flow velocity into the domain $v_0 = -\mathbf{n} \bullet \mathbf{u}_0$, the upstream density $\rho_0$, and the relative species mole fractions $\mathbf{X}_0$. The weak form of the FE discretization yields a surface integral of the diffusive flux over the inlet boundary. Using (A.4) to solve for the diffusive flux, we have

$$\mathbf{n} \bullet \mathbf{j}^i_{diffusive} = \mathbf{n} \bullet \left( \mathbf{j}^i_{total} - \mathbf{j}^i_{convective} \right) = -\rho_0 v_0 Y^i_0 + \rho v Y^i , \tag{A.5}$$

where $Y_0^i$ is the mass fraction of species $i$ computed from the given mole fractions $\mathbf{X}_0$, $Y^i$ is the unknown mass fraction of species $i$ at the inlet boundary, $v = -\mathbf{n} \bullet \mathbf{u}$ is the unknown normal velocity into the domain at the inlet boundary, and $\rho$ and $\rho_0$ are the densities calculated for $\mathbf{Y}$ and $\mathbf{Y}_0$. By conservation of mass, the total mass flux of species $i$ at the inlet boundary must be equal to the given mass flux into the system,

$$\rho v = \rho_0 v_0, \tag{A.6}$$

which leads to a Dirichlet condition on the inlet velocity:

$$v = \rho_0 v_0 / \rho. \tag{A.7}$$

Using (A.7) to simplify (A.5), we get a MIXED boundary condition for each species,

$$\mathbf{n} \bullet \mathbf{j}^i_{diffusive} = \rho_0 v_0 (Y - Y_0). \tag{A.8}$$

With MPSalsa, (A.7) and (A.8) are applied with the following lines in the Boundary Condition section of the input file (assuming that the boundary is side set 1 and has a normal in the $y$-direction):

```
BC = V_BC DIRICHLET SS 1 DEPENDENT f_Danckwerts_X0 1
      BC_DATA = FLOAT S_0  X1 X2 X3 .. XN
BC = Y_BC MIXED SS 1 INDEPENDENT f_Danckwerts f_Danckwerts_X0 0.0 0.0 1
      SPECIES_LIST = ALL
      BC_DATA = FLOAT S_0  X1 X2 X3 .. XN
```

The BC_DATA statements following the V_BC and Y_BC statements must be the same, and consist of an upstream velocity S_0 followed by the list of molar flux fractions. The expression for S_0 varies depending on the type of velocity boundary condition in which it is used. For Dirichlet boundary conditions on one component $u_k$ of the velocity $\mathbf{u}$ (i.e., U_BC, V_BC, or W_BC),

$$\text{S\_0} = v_0 (-\mathbf{n} \bullet \mathbf{e}_k), \tag{A.9}$$

where $\mathbf{e}_k$ is the unit vector in the $k^{th}$-coordinate direction. For Dirichlet boundary conditions on the normal velocity $\mathbf{n} \bullet \mathbf{u}$ (i.e., VEL_NORM_BC),

$$\text{S\_0} = \mathbf{n} \bullet \mathbf{u}_0 = -v_0. \tag{A.10}$$

In both of these cases, S_0 is the velocity value that would be used if regular Dirichlet boundary conditions on velocity were being imposed instead of Danckwerts' boundary condition.

The function f_Danckwerts_X0, when used as a velocity boundary condition, calculates the ratio of the densities in (A.7) and multiplies it by S_0. When used as a Y_BC, this function returns the appropriate mass fraction calculated from the mole fractions $X_1, X_2, \ldots, X_N$. The function f_Danckwerts returns the quantity $\rho_0 v_0 = \rho_0 |\mathtt{S\_0}|$, which is analogous to the heat transfer coefficient in the typical MIXED boundary condition. It calculates $\rho_0$ assuming that the temperature and pressure upstream of the boundary are equal to those values used at the boundary. Thus, only the mass fractions and the normal velocity are allowed to have a jump discontinuity between the upstream and the domain. This limits effective usage of this boundary condition to cases where there is a Dirichlet condition on the temperature on the same boundary.

If the inlet fluxes are known in terms of mass fractions instead of mole fractions, the function f_Danckwerts_Y0 can be used in place of the f_Danckwerts_X0 above, and the list of mass fractions must follow S_0 in the BC_DATA statements.

### A.1.3. Spinning Disk Boundary Conditions

#### A.1.3.1. Spinning Disk in the *xy*-Plane

The boundary condition function f_xy_spin_disk is used to apply Dirichlet boundary conditions on velocities on a spinning disk in the *xy*-plane. This function returns non-zero values only for boundary condition types U_BC and V_BC. It should be called as an independent Dirichlet condition on either side sets or node sets, and requires a BC_DATA statement. The BC_DATA line must include three floating point numbers, the first being the disk rotation rate in rpms (revolutions per minute) in the counterclockwise direction. The next two entries are the coordinates of the rotation center.

For example, boundary conditions for a disk rotating at 80 rpm that is centered at the point $(x,y) = (2,-3)$ would be imposed using the following lines in the input file:

```
U_BC DIRICHLET NS 1 INDEPENDENT f_xy_spin_disk 1
     BC_DATA = 80.0 2.0 -3.0
V_BC DIRICHLET NS 1 INDEPENDENT f_xy_spin_disk 1
     BC_DATA = 80.0 2.0 -3.0
```

The rotation rate is translated from rpm to radians/sec in a pre-processing step in the file "rf_input_bc.c."

#### A.1.3.2. Spinning Tilted Disk

The boundary condition f_xy_spin_tilt9_disk was written for the Tilted CVD reactor (see the example in Appendix D.3). In this reactor, the rotating substrate is on a tilted plane whose tangent vectors are $(1, 0, 0)$ and $(0, \cos\varphi, \sin\varphi)$, with $\varphi = 9$ degrees. Since the velocity normal to the disk can be non-zero due to the Stefan velocity, the rotation boundary conditions are imposed in the two tangential directions using the Generalized Surface functionality.

As with the spinning disk boundary condition in Appendix A.1.3.1, this independent Dirichlet condition requires a BC_DATA statement with the rotation rate, followed by the center of rotation. An example using this boundary condition, including the specification of the generalized surface along side set 5, is shown in Figure A.2. This specification is for a disk centered at $(0, 0, 1.5046)$ that is rotating at 80 rpm.

```
Number of Generalized Surfaces = 1
GENERALIZED_SURFACE 5 2
        TANGENT 1.0 0.0 0.0
        TANGENT 0.0 0.9876 0.1564

Number of BC = 33
BC = VEL_TAN1_BC DIRICHLET GS 1 INDEPENDENT f_xy_spin_tilt9_disk 1
        BC_DATA = FLOAT 80.0 0.0 0.0 1.5046
BC = VEL_TAN2_BC DIRICHLET GS 1 INDEPENDENT f_xy_spin_tilt9_disk 1
        BC_DATA = FLOAT 80.0 0.0 0.0 1.5046
...
```

*Figure A.2. Example usage of* `f_xy_spin_tilt9_disk` *to specify Dirichlet boundary conditions for velocities on a spinning, tilted disk.*

## A.1.4. Mass Fraction Dirichlet Boundary Conditions expressed as Mole Fractions

In MPSalsa, the primitive variables for mass transfer are mass fractions, but for many applications, it is the mole fractions that are known. MPSalsa includes the function `f_mole_fraction` which allows the user to specify the mole fractions as a Dirichlet condition along a side set or node set. An example of this boundary condition is in Figure A.3. The mole fractions for all species are listed on the BC_DATA line in the order of the SPECIES_LIST arguments above it. For SPECIES_LIST = ALL, the mole fractions should be listed in order from the first species to the last species. The mole fractions can be spread across more than one BC_DATA statement, each preceded by a SPECIES_LIST statement.

```
BC = Y_BC DIRICHLET SS 1 INDEPENDENT f_mole_fraction 1
SPECIES_LIST = 2 1 4 3
BC_DATA = 1.232900e-04 1.095458e-02 9.889221e-01 0.0
```

*Figure A.3. Example usage of* `f_mole_fraction` *to specify Dirichlet boundary conditions for mass fractions in terms of mole fractions.*

The conversion from mole fractions to mass fractions is done once in a preprocessing step, with the resulting mass fractions being stored in the `BC_Data_Float` array where the mole fractions originally were. Error checking makes sure that each species is assigned a mole fraction and that the sum of mole fractions is near unity.

### A.1.5. Outflow Boundary Condition

The included function `f_pressure` returns the hydrodynamic pressure unknown weighted by a constant. This value can be used as an outflow boundary condition by imposing this function as a Neumann condition on the normal component of the momentum equation. The usage of the function in the case of outflow from the computational domain on side set 3, with a normal in the *y*-direction, is

```
    BC = V_BC NEUMANN SS 3 DEPENDENT f_pressure 1
    BC_DATA = FLOAT 1.0
```

The single floating point data statement required with the `f_pressure` boundary condition is a multiplicative factor, which will be discussed later.

A reasonable outflow boundary condition on the normal component of the momentum balance is that the normal velocity is not changing as it leaves the domain, i.e. $du_n/dn = 0$ where **n** represents the direction normal to the boundary and $u_n$ is the normal velocity. The weak form of the FE residual equation in the direction normal to the surface with respect to test function $\Psi_j$ renders the following surface integral for the normal component of the stress tensor:

$$\int_{\Gamma}\left(-P-\frac{2}{3}\mu\nabla\bullet\mathbf{u}+2\mu\frac{du_n}{dn}\right)\Psi_j d\Gamma . \qquad (A.11)$$

From the continuity equation, the middle term is identically zero for incompressible flows and is often negligible for variable-density flows. A natural condition that sets the entire integral to zero works for many cases as an outflow boundary condition and has the added feature of setting the pressure datum to near zero along the outflow surface. Thus, no boundary condition for pressure is needed for open flows while the pressure must be set at one node for closed flows.

This natural condition does not work for cases where the pressure is not constant along the outflow surface, such as a vertical outflow plane in systems with gravity and swirling flows such as the Rotating Disk Reactor configuration in Appendix D.2. It is for these systems that we impose the simple `f_pressure` Neumann boundary condition

$$k\int_{\Gamma}(-P)\,\Psi_j d\Gamma\,. \qquad \text{(A.12)}$$

In (A.12), $k$ is the multiplicative floating point number input in the BC_DATA statement. When $k = 1.0$ and the divergence of the velocity is negligible, this boundary condition weakly imposes the desired outflow boundary condition. However, this results in an arbitrary pressure datum again. We have found empirically that setting the multiplicative constant in the range of $k \in [0.9, 0.99]$ gives smooth outflow profiles while still setting the average pressure on the outflow boundary to zero.

The FIDAP package [13] also integrates the pressure as an outflow boundary condition, but does not include the derivatives of the boundary condition in the Jacobian matrix. The pressure from the previous Newton iteration sets the pressure at the current step, removing the need for a value of $k$ other than unity to set the pressure datum. However, this omission can greatly degrade convergence of Newton's method. The user can try this method by changing the boundary condition to type INDEPENDENT so that no Jacobian entries are computed for this boundary condition. Other outflow boundary conditions are under development.

## A.2. Look-up Tables

Values of properties and boundary conditions may be interpolated from tables of data specified in the Function Data section of the input file (see Section 3.11). Two of these look-up tables, lookup_table_1 and lookup_table_2, are included in MPSalsa. Other look-up tables can be easily added by following the example of lookup_table_1 in "rf_fn_data.c" (actual code for the function), "rf_fill_const.h" (prototype for the function), and "rf_bc_exact_fn.c" and "rf_source_fn.c" (pointer assignment routines for the function).

Look-up tables can be used anywhere a SNGLVAR_FUNCTION can be used (see Table 4.2). For example, to use a look-up table to compute the volumetric source term as a function of temperature for the mass fraction equations, a variable mass fraction source term is specified in the Material Properties section of the input file (see Section 3.6):

```
    Y_VOLUME_VAR = lookup_table_1 single
```

The data for lookup_table_1 is included as a TABLE in the Function Data section of the input file:

```
Function = lookup_table_1 2
FN_DATA = STRING TEMPERATURE
FN_DATA = TABLE n 2
      t1  q1
      t2  q2
       ...
      tn  qn
```

where $t_1, t_2, \ldots, t_n$ are the values of the temperatures (in increasing order) and $q_1, q_2, \ldots, q_n$ are corresponding mass fraction source term values. The `FN_DATA STRING` indicates the independent variable to use in the table. The look-up function uses linear interpolation to compute the source term using the values of the independent variable passed to `lookup_table_1`.

## A.3. Output

The following functions have been written to provide some useful output from MPSalsa for the analysis of solutions. Still, the majority of post-processing is left to graphics packages that can read ExodusII files.

None of the following functions are called automatically from MPSalsa, but must be explicitly called from the function `user_out` in the file "rf_user_out.c." The function calls and argument lists are described in comments at the top of each function.

The `status` variable, described in Figure 4.9, can be used to restrict the output. For instance, the function call can be preceded by the following condition if output is not desired for failed time steps:

```
if (status >=0).
```

### A.3.1. Evolution of the Solution at a Point

The evolution of the solution at a point (or points) in the domain can be output from MPSalsa using the `time_history_points` output function. Two things must be done to use this function. First, the function call

```
time_history_points(time, time_step_num, soln);
```

must be added to the function `user_out` in the file "rf_user_out.c" and the code must be recompiled. Second, data must be input for this function in the Function Data Specifications section of the input file (see Section 3.11). This function needs only a list of points at which the solution output is desired. For instance, the following section of input file

```
Function Name = time_history_points 1
FN_DATA = TABLE 2 3
    0.0 0.01 0.5
    0.0 0.99 0.5
```

would cause the entire solution at $(0, 0.01, 0.5)$ to be printed at each time step to the file "time_his.0," and the solution $(0, 0.99, 0.5)$ to be printed to the file "time_his.1." The two integers following the `TABLE` keyword specify the dimensions of the table to be read, with the first number (2) representing the number of points at which to print data and the second number (3) specifying the dimension of the system.

Each line of the output file contains the following information: time step number, time, $x$, $y$, $z$ (for 3D problems), and the entire solution at the point (with mass fractions translated to mole fractions), in the following order: $\mathbf{u}$, $P$, $T$, $X_1, X_2, \ldots, X_N$. This output format allows for easy plotting with a package such as "gnuplot," where plotting column 7 versus column 3 gives a plot of $y$-velocity $u_2$ versus time.

## A.3.2. The Solution along a Line

The `time_history_line` output function gives the ability to analyze the solution along a line through the computational domain. This function has been used to generate many of the plots in the example problems shown in subsequent appendices.

The implementation of this function is almost identical to `time_history_points`. A call to the function

```
time_history_line(time, time_step_num, soln);
```

must be included in `user_out` and the code must be recompiled. The `status` flag can be used to restrict some output, as described in Figure 4.9.

In the Function Data Specifications section of the input file (see Section 3.11), data must be entered for this function. Two data lines are required: an integer that tells how many points on the line are desired, and a table with two rows that gives the beginning and ending points of the line. Solutions along more than one line can be output by supplying more than one set of data to the function. The input lines in Figure A.4 show how this is done for a 2D problem. One line gives a slice through the domain as a function of $x$, and the other is a slice in the $y$-direction. Each line is written to a separate file and, unlike the `time_history_points` function, the data at each time step is written to a separate file. For instance, with the input data in Figure A.4, the solution at the 80 points equally spaced on the line between $(0,0)$ and $(1,0)$ at the 14th time step will be in the file "time_his_line.0.14," and the 50 points equally spaced between $(0.5,-10.0)$ and $(0.5,10.0)$ at the 7th time step will be in the file "time_his_line.1.7."

```
Function Name   =  time_history_line 2
FN_DATA = INT 80
FN_DATA = TABLE  2 2
       0.0 0.0
       1.0 0.0
# Second line for time history output:
Function Name   =  time_history_line 2
FN_DATA = INT 50
FN_DATA = TABLE  2 2
       0.5 -10.0
       0.5 10.0
```

*Figure A.4. Example function data lines for* `time_history_line`.

As with the `time_history_points` function, each line of the output file contains the following information: time step number, time, *x*, *y*, *z* (for 3D problems), and the entire solution at the point (with mass fractions translated to mole fractions), in the following order: $\mathbf{u}$, $P$, $T$, $X_1, X_2, \ldots, X_N$.

### A.3.3. Information on a Side Set

The function `f_ss_centroid` gives the user the ability to print many useful pieces of information along a side set. Information from this function can be used to get such information as the average temperature on a surface, the total heat flux into a wall, and the drag coefficient over a body. The function calculates positions, solution values, normal gradients, and other information at the centroid of the surface elements in one or more side sets.

The implementation of this routine requires that the following function call be added as one of the first executable statements of function `user_out`:

```
    f_ss_centroid(time, time_step_num, soln);
```

The code then must be recompiled. Also, data must be given to this function in the Function Data Specifications section of the input file (see Section 3.11). An example is given here.

```
    Function Name = f_ss_centroid 2
    FN_DATA = INT 1 2 3
    FN_DATA = STRING x T Area
```

The required integer data is a list of side set IDs for which information is to be printed. In this case, information will be output for side sets 1, 2, and 3 all to the same output file. If it is desired that the data be separated into different files for each side set, multiple sets of data can be supplied to this function (with repeated `Function` Name lines), each with a single integer for the side set list.

The STRING data specifies the quantities to be output. In this example, the $x$-coordinate, the temperature, and the area (length) of the surface element are output. Table A.1 lists the strings currently recognized by this function and the quantity that each string refers to. In the future, we hope to add physical quantities such as the local density or viscosity to the list of recognized strings.

| STRING | OUTPUT |
|---|---|
| `t, time` | Time value |
| `x` | $x$-coordinate of position |
| `y` | $y$-coordinate of position |
| `z` | $z$-coordinate of position |
| `U` | Velocity in the $x$ direction |
| `V` | Velocity in the $y$ direction |
| `W` | Velocity in the $z$ direction |
| `P` | Hydrodynamic pressure |
| `T` | Temperature |
| **`Y`** | Array of mass fractions |
| `A, Area` | Area (length) of the element |
| **`n, normal`** | Outward pointing normal vector |
| **`t1, tangent`** | Tangent vector |
| **`t2, tangent2`** | Second tangent vector (for 3D problems) |
| `Vn, Un` | Velocity in the normal direction |
| `n_grad_U` | Normal component of the gradient of the $x$-component of velocity |
| `n_grad_V` | Normal component of the gradient of the $y$-component of velcocity |
| `n_grad_W` | Normal component of the gradient of the $z$-component of velocity |
| `n_grad_P` | Normal component of the gradient of $P$ |
| `n_grad_T` | Normal component of the gradient of $T$ |
| **`n_grad_Y`** | Normal component of the gradient of $Y_i$, for all $i$ |
| **`tau_n`** | Traction vector / viscosity, no pressure contribution |

*Table A.1. List of Strings currently recognized by the* `f_ss_centroid` *output function. The bold strings lead to more than one column of output.*

The output from this function is written to files of the form "ss_data.*n.m*" where the integer $n$ identifies the set of function data ($n = 0$ for the first occurrence of `f_ss_centroid`,

$n = 1$ for the second occurrence, etc.), and $m$ is the time step number. Each file has one line for each element in the side set(s), and each line has at least one column for each quantity specified in the `STRING` data statement.

Integrated quantities over the side set can be calculated using the element area information. For instance, the total conductive heat flux through the side set can be calculated by summing over all surfaces in the side set the products of the area (`A`) of each surface with the normal gradient of the temperature (`n_grad_T`) and the thermal conductivity. Averages can be computed by summing over all surfaces the product of a quantity with the surface's area, and dividing the sum by the total area.

The `tau_n` string leads to an array of output that includes the components of the viscous traction vector along the surface:

$$\texttt{tau\_n} = -\frac{2}{3}\nabla\bullet\mathbf{u} + 2\frac{du_n}{dn}. \tag{A.13}$$

Note that `tau_n` does not include the pressure term, which can be output independently, and does not include the multiplication by the viscosity. The total drag force over an object in the $x$-direction is the sum of the first component of `tau_n` (`tau_x`) multiplied by the viscosity and the element area (`A`).

## A.4. Interprocessor Communication Utilities

This section details some machine-independent communication functions callable within MPSalsa that are useful when programming new functions for parallel applications, especially when I/O is involved. The code for these functions is in "rf_comm.c."

### A.4.1. Synchronization

Certain operations require that all processors are at the same part of the code at the same time. A call to the `sync` function causes each processor to wait until all processors have reached the statement. The syntax is

```
    sync(Proc, Num_Proc);
```

where `Num_Proc` is the total number of processors running the problem, and `Proc` is the unique processor ID with a value between 0 and (`Num_Proc - 1`) of the current processor. Both `Num_Proc` and `Proc` are defined as global integer variables in MPSalsa and are initialized at the beginning of MPSalsa's execution. If any processor fails to reach the `sync` statement, the computation will idle indefinitely.

When each processor must write to a common output file, the print statement should be surrounded by the `print_sync_start` and `print_sync_end` function calls. These functions synchronize the processors so that only one processor at a time executes the statements between the calls. There can be no communication calls between these statements; such calls would cause the program to reach a deadlocked state.

The code fragment in Figure A.5 demonstrates the use of `print_sync_start` and `print_sync_end`. The resulting output file would contain the processor ID numbers printed in order from 0 to `Num_Proc` - 1.

```
print_sync_start(Proc, Num_Proc,1);
   if (Proc==0) ifp = fopen("filename","w");
   else ifp = fopen("filename","a");
   fprintf(ifp,"%d \n",Proc);
   fclose(ifp);
print_sync_end(Proc, Num_Proc);
```

*Figure A.5. Code fragment demonstrating the use of* `print_sync_start` *and* `print_sync_end`.

## A.4.2. Broadcast

A machine-independent broadcast routine called `brdcst` has been written for use in MPSalsa. Information on one processor (usually processor zero) is sent to all other processors using this routine. There are five arguments for this function; the first two are `Proc` and `Num_Proc`; the third is the pointer to the memory location where the information is stored or to be stored; the fourth is the message size; and the last is the number of the processor that is initiating the broadcast (usually processor zero).

The code fragment in Figure A.6 illustrates the use of this routine, by broadcasting an array of length two from processor zero to all other processors. The message size is the array length (two) times the size of a double variable (computed using the `sizeof` function).

```
double x[2];

if (Proc==0) {
   x[0] = 10.5;
   x[1] = 0.123;
}
brdcst(Proc, Num_Proc, (char *) x, 2*sizeof(double), 0);
```

*Figure A.6. Code fragment demonstrating the use of* `brdcst`. *Upon return from* `brdcst`, *x=[10.5, 0.123] on all processors.*

### A.4.3. Global Sum, Maximum, and Minimum

Several functions that compute the sum, maximum or minimum of some value over all processors are included in MPSalsa. Several of these functions are listed in Figure A.7. The functions `gsum_int`, `gmax_int`, and `gmin_int` compute the sum, maximum and minimum, respectively, of an integer value. The functions `gsum_double`, `gmax_double`, and `gmin_double`, perform the same operations on double precision variables. In all cases, the first argument is the quantity that is to be summed or compared.

```
   int i,j;
   double x,y;
...
   j = gsum_int    (i, Proc, Num_Proc);
   j = gmax_int    (i, Proc, Num_Proc);
   j = gmin_int    (i, Proc, Num_Proc);
   y = gsum_double (x, Proc, Num_Proc);
   y = gmax_double (x, Proc, Num_Proc);
   y = gmin_double (x, Proc, Num_Proc);
```

*Figure A.7. Functions for computing the sum, maximum and minimum of a value over all processors. The functions* `gsum_int`, `gmax_int` *and* `gmin_int` *operate on integers; the functions* `gsum_double`, `gmax_double`, *and* `gmin_double` *operate on double precision variables.*

## Appendix B. Mass Transfer Examples

### B.1. Diffusion in an Annulus

This simple example problem consists of a single species diffusing in an annular region, and is designed to illustrate the use of the three different boundary condition types: Dirichlet, Neumann, and Mixed. The domain has inner radius of $R_i = 1$ and an outer radius of $R_o = 2$. The domain is discretized with the 2048 element mesh shown Figure B.1, with the inner circle designated Side Set 1 and the outer circle designated Side Set 2.

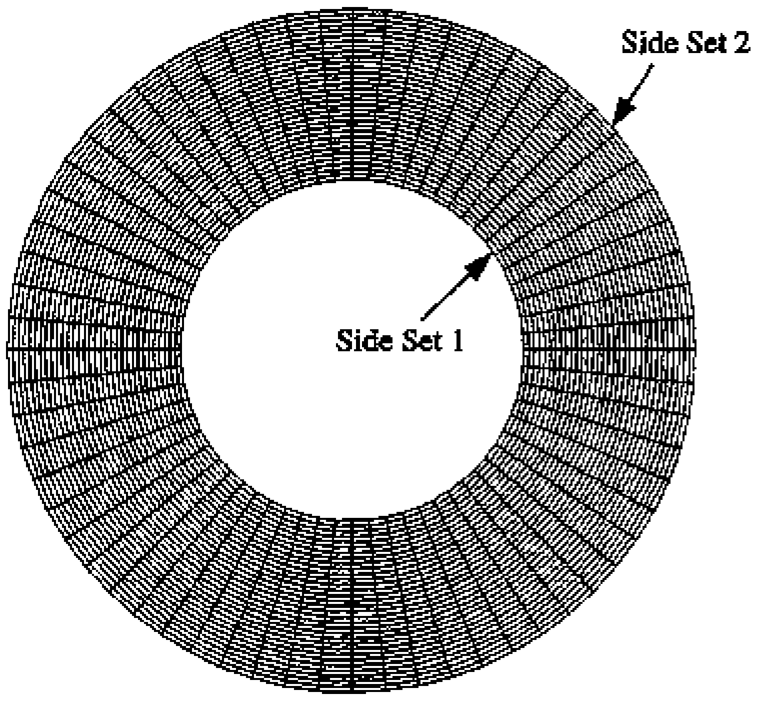


*Figure B.1. Finite element mesh for the* Diffusion in an Annulus *example problem. The mesh contains 2048 elements and 2112 nodes and is stored in the file* washer.exoII.

A volumetric mass source of magnitude one generates mass uniformly over the domain, and the diffusion coefficient is also set equal to unity, leading to the following governing equation:

$$\nabla^2 C + 1 = 0, \qquad \text{(B.1)}$$

where C is a dimensionless concentration. At the inner circle of the annulus, we set a Dirichlet condition of

$$C = 1 \text{ for } r = \sqrt{x^2 + y^2} = R_i. \tag{B.2}$$

To illustrate the three different standard boundary condition types available in MPSalsa, we pose three options for the boundary condition at the outer circle (Side Set 2):

either Dirichlet:

$$C = 1/4 \text{ for } r = R_o; \tag{B.3}$$

or Neumann:

$$\mathbf{n} \cdot \nabla C = 1 \text{ for } r = R_o; \tag{B.4}$$

or Mixed (Robin):

$$\mathbf{n} \cdot \nabla C = 4\,(C - 0) \text{ for } r = R_o. \tag{B.5}$$

Any of these three boundary conditions leads to the same analytic solution:

$$C = \frac{5 - x^2 - y^2}{4}. \tag{B.6}$$

This function has been programmed into a function called `f_annulus_exact` to test the computed solution.

The MPSalsa input file for solving this problem is given in Figure B.2. It shows that we are solving a diffusion-only problem to a steady-state solution using the GMRES method with preconditioning. The number of species and the volumetric source term are set in the Materials Specifications section. At the end of the Output Specifications section, it is specified that the final solution be tested against the analytic solution programmed in `f_annulus_exact`. As can be seen in the Boundary Conditions section, this file applies the Dirichlet condition (B.3) on Side Set 2. The options of applying the Neumann condition (B.4) or Mixed condition (B.5) are commented out by the pound sign (#).

Table B.1 compares the solutions for the three boundary condition types, by showing the $L^2$-error of the computed solution with respect to the analytic solution, the CPU time on an SGI workstation needed to reach the solution, and the number of GMRES linear solve iterations needed to reach the solution. Since the problem is linear, each solution required only one Newton iteration. There is no significant difference between the three solutions, except the Neumann case required a few more linear iterations.

```
----------------------------------------------------------------
              General Problem Specifications
----------------------------------------------------------------
Problem type                        = mass_diff
Input FEM file                      = Meshes/washer.exoII
LB file                             = none
Output FEM file                     = run_out.exoII
Number of processors                = 1
Cartesian or Cylindrical when 2D    = Cartesian
Interpolation Order                 = linear
Stabilization                       = default
Debug                               = 2
----------------------------------------------------------------
                 Solution Specifications
----------------------------------------------------------------
Solution type                           = steady
Order of integration/continuation       = 1
Step Control                            = off
Relative Time Integration Error         = 1.0e-3
Initial Parameter Value                 = 100.0
Initial Step Size                       = 2.0e-1
Maximum Number of Steps                 = 1000
Maximum Time or Parameter Value         = 250
----------------------------------------------------------------
                 Solver Specifications
----------------------------------------------------------------
Override Default Linearity Choice                   = default

-- nonlinear solver subsection

Number of Newton Iterations                         = 15
Use Modified Newton Iteration                       = no
Enable backtracking for residual reduction          = no
Choice for Inexact Newton Forcing Term              = 4
Calculate the Jacobian Numerically                  = no
Solution Relative Error Tolerance                   = 1.0e-3
Solution Absolute Error Tolerance                   = 1.0e-8

-- linear solver subsection

Solution Algorithm                                  = gmres
Convergence Norm                                    = 1
Preconditioner                                      = no_overlap_ilu
Polynomial                                          = LS,1
Scaling                                             = row_sum
Orthogonalization                                   = classical
Size of Krylov subspace                             = 25
Maximum Linear Solve Iterations                     = 50
Linear Solver Normalized Residual Tolerance         = 1.0e-6
----------------------------------------------------------------
                 Chemistry Specifications
----------------------------------------------------------------
Energy  equation source terms           = off
Species equation source terms           = off
Pressure (atmospheres)                  = 1.0
Thermal Diffusion                       = off
Multicomponent Transport                = stefan_maxwell
Chemkin file                            = chem.bin
Surface chemkin file                    = surf.bin
Transport chemkin file                  = tran.bin
----------------------------------------------------------------
           Enclosure Radiation Specifications
----------------------------------------------------------------
Enclosure Radiation source terms        = off
----------------------------------------------------------------
          Material ID Specifications
----------------------------------------------------------------
Number of Materials                     = 1
SOLID                        = 0     "Graphite"
        ELEM_BLOCK_IDS                  = 1
        NUM_SPECIES                     = 1
        SPECIES_NAME          1         YK_1
        DIFF_COEFF YK_1 1.0
        WTSPECIES  YK_1 1.0
        XMF_0      YK_1 1.0
# Source Term
        Y_VOLUME  =  1.0
END Material ID Specifications
```

```
----------------------------------------------------------------
         Boundary Condition Specifications
----------------------------------------------------------------
Number of Generalized Surfaces = 0
Number of BC                   = 2
# BC on inner radius, r=1
BC = Y_BC DIRICHLET SS 1 INDEPENDENT 1.0  0
     SPECIES_LIST = ALL
#
# BC on outer radius, r=2
#Dirichlet
BC = Y_BC DIRICHLET SS 2 INDEPENDENT 0.25 0
#Neumann
#BC = Y_BC NEUMANN SS 2 INDEPENDENT  1.0  0
#Mixed
#BC = Y_BC MIXED SS 2 INDEPENDENT 4 0 0 0 0 0 0.0  0
     SPECIES_LIST = ALL
----------------------------------------------------------------
         Initial Guess/Condition Specifications
----------------------------------------------------------------
Set Initial Condition/Guess                 = constant 0.0
Apply function                              = no
Time Index to Restart From                  = 1
----------------------------------------------------------------
                 Output Specifications
----------------------------------------------------------------
User Defined Output                                  = yes
Parallel Output                                      = no
Scalar Output                                        = yes
Time Index to Output To                              = 1
Nodal variable output times
        every 1 steps

Number of nodal output variables                     = 1
Nodal variable names
        Mass_fraction

Number of global output variables                    = 0
Global variable names

Test Exact Solution Flag                             = 1
Name of Exact Solution Function      = f_annulus_exact
----------------------------------------------------------------
                 Parallel I/O section
----------------------------------------------------------------
Machine                                     = paragon
Staged writes                               = yes

----------------
ncube subsection
----------------

Number of controllers                       = 8
Disks per controller                        = 1
Root location                               = //df
Subdirectory                                = jns/tests
Offset numbering from zero                  = 0

-----------------
paragon subsection
-----------------

Number of RAID controllers                  = 26
Root location                               = /pfs/io_
Subdirectory                                = tmp/kdd/t143
Offset numbering from zero                  = 23
----------------------------------------------------------------
       Data Specification for User's Functions
----------------------------------------------------------------
Number of functions to pass data to    =  0
```

*Figure B.2. Input file for the* Diffusion in an Annulus *example problem.*

| BC Type on Side Set 2 | $L^2$-Error | CPU Time (seconds) | Number of GMRES Iterations |
|---|---|---|---|
| Dirichlet | 2.20e-4 | 1.16 | 16 |
| Neumann | 1.75e-4 | 1.34 | 23 |
| Mixed | 2.12e-4 | 1.19 | 17 |

*Table B.1. Comparison of the three boundary condition types for the* Diffusion in an Annulus *example problem.*

## B.2. The Soret Effect

This simple example of thermal diffusion (the Soret effect) illustrates the use of a CHEMKIN material type. The problem is solved on a 2D mesh but is essentially 1D. Hydrogen (H2 -- molecular weight 2.016) and Trimethylgallium (GaMe3 -- molecular weight 114.83) are allowed to interdiffuse along a steep thermal gradient. The 100-element mesh and boundary conditions are shown in Figure B.3.

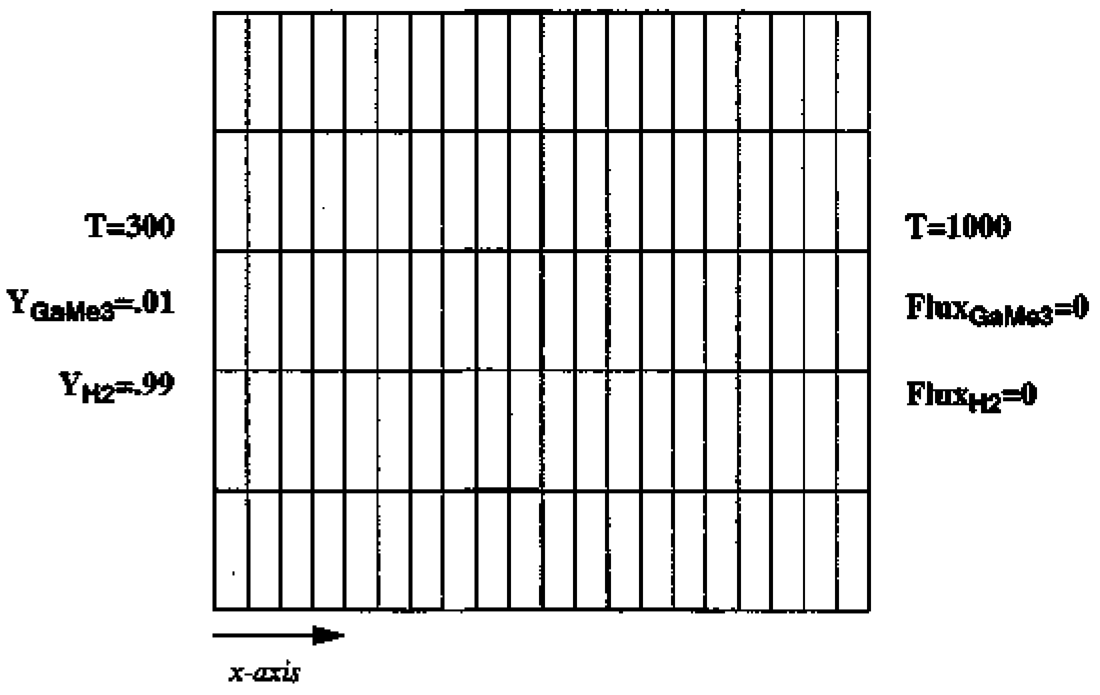


*Figure B.3. 100 element mesh and boundary conditions for the* Soret Effect *example problem.*

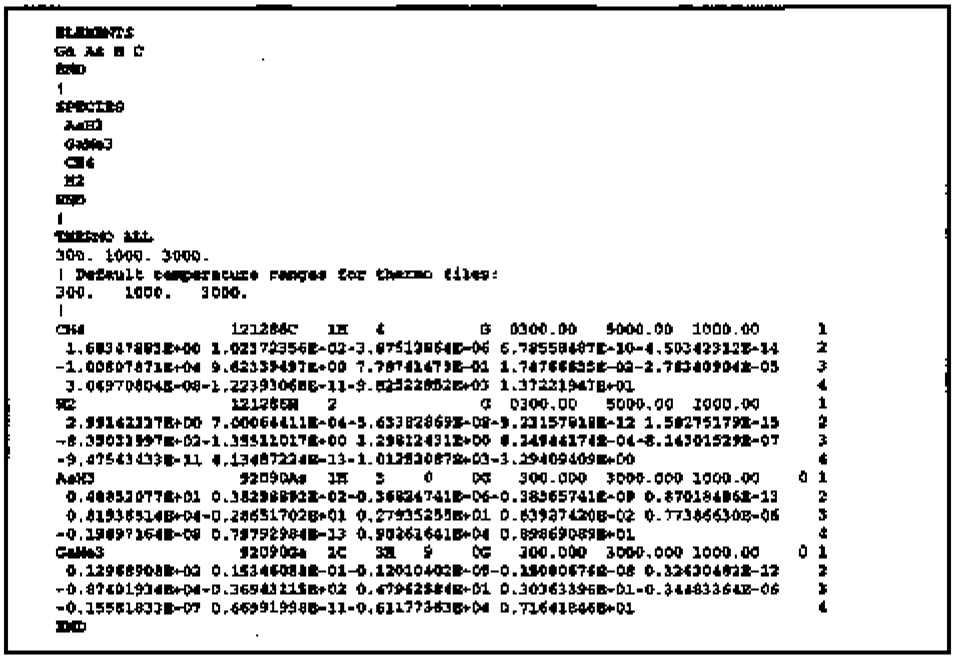
The input file for this example problem is shown in Figure B.4, and shows that this is an energy and mass transfer problem, being solved directly to the steady-state using GMRES and a preconditioner. Because the material is a CHEMKIN material, the number of species, species names, molecular weights, and transport properties are not specified in the Materials Specifications section. This information is read into MPSalsa from the file "chem.bin," which is

```
-----------------------------------------------------------
              General Problem Specifications
-----------------------------------------------------------
Problem type                        = energy_mass_diff
Input FEM file                      = Meshes/box100 exoII
LB file                             = none
Output FEM file                     = run-out exoII
Number of processors                = 1
Cartesian or Cylindrical when 2D    = Cartesian
Interpolation Order                 = linear
Stabilization                       = default
Debug                               = 2
-----------------------------------------------------------
                Solution Specifications
-----------------------------------------------------------
Solution Type                           = steady
Order of integration/continuation       = 1
Step Control                            = on
Relative Time Integration Error         = 4 0e-3
Initial Parameter Value                 = 300 0
Initial Step Size                       = 1 0e-3
Maximum Number of Steps                 = 10
Maximum Time or Parameter Value         = 1 0e+9
-----------------------------------------------------------
                  Solver Specifications
-----------------------------------------------------------
Override Default Linearity Choice                 = default

----------------- nonlinear solver subsection  -------------

Number of Newton Iterations                       = 25
Use Modified Newton Iteration                     = no
Enable backtracking for residual reduction        = no
Choice for Inexact Newton Forcing Term            = 4
Calculate the Jacobian Numerically                = no
Solution Relative Error Tolerance                 = 1 0e-3
Solution Absolute Error Tolerance                 = 1 0e-8

----------------- linear solver subsection  ---------------

Solution Algorithm                                = gmres
Convergence Norm                                  = 1
Preconditioner                                    = no_overlap_ilu
Polynomial                                        = LS,1
Scaling                                           = row_sum
Orthogonalization                                 = classical
Size of Krylov subspace                           = 100
Maximum Linear Solve Iterations                   = 200
Linear Solver Normalized Residual Tolerance       = 1 0e-6
-----------------------------------------------------------
                 Chemistry Specifications
-----------------------------------------------------------
Energy  equation source terms          = off
Species equation source terms          = off
Pressure (atmospheres)                 = 0 1
Thermal Diffusion                      = on
Multicomponent Transport               = stefan_maxwell
Chemkin file                           = chem bin
Surface chemkin file                   = surf bin
Transport chemkin file                 = tran bin
-----------------------------------------------------------
           Enclosure Radiation Specifications
-----------------------------------------------------------
Enclosure Radiation source terms      = off
-----------------------------------------------------------
        Material ID Specifications
-----------------------------------------------------------
Number of Materials               = 1
CHEMKIN                           = 0  "gas_block"
     ELEM_BLOCK_IDS = 1
     T_INIT         = 500

     XMF_0 GaMe3     0 01
     XMF_0 H2        0 99

END Material ID Specifications
```

```
-----------------------------------------------------------
        Boundary Condition Specifications
-----------------------------------------------------------
Number of Generalized Surfaces = 0
Number of BC                     = 3
BC    = T_BC DIRICHLET SS 4 INDEPENDENT  300 0 0
BC    = T_BC DIRICHLET SS 3 INDEPENDENT 1000 0 0
BC    = Y_BC DIRICHLET SS 4 INDEPENDENT f_mole_fraction 1
        SPECIES_LIST =  H2 GaMe3 AsH3 CH4
        BC_DATA = FLOAT  0 99 0 01 0 0 0 0
-----------------------------------------------------------
        Initial Guess/Condition Specifications
-----------------------------------------------------------
Set Initial Condition/Guess                 = constant 0 0
Apply function                              = no
Time Index to Restart From                  = 1
-----------------------------------------------------------
                Output Specifications
-----------------------------------------------------------
User Defined Output                                  = yes
Parallel Output                                      = no
Scalar Output                                        = yes
Time Index to Output To                              = 1
Model variable output times
        every 2 steps

Number of nodal output variables                     = 2
Nodal variable names
        Temperature
        Mass_fraction

Number of global output variables                    = 0
Global variable names

Test Exact Solution Flag                             = 0
Name of Exact Solution Function                      = none
-----------------------------------------------------------
                Parallel I/O section
-----------------------------------------------------------
Machine                                        = paragon
Staged writes                                  = yes

----------------
ncube subsection
----------------

Number of controllers                          = 8
Disks per controller                           = 1
Root location                                  = //df
Subdirectory                                   = jns/tests
Offset numbering from zero                     = 0

----------------
paragon subsection
-----------------

Number of RAID controllers                     = 19
Root location                                  = /pfs/io_
Subdirectory                                   = tmp/ags/em
Offset numbering from zero                     = 2
-----------------------------------------------------------
      Data Specification for User's Functions
-----------------------------------------------------------
Number of functions to pass data to   =  1

Function Name    =  time_history_line 2
$
FN_DATA = INT  25
FN_DATA = TABLE  2  2
      0 0 0 5
      1 0 0 5
```

*Figure B.4. Input file for the* Soret Effect *example problem.*

generated using the "interp" utility acting on the Chemkin input file for gas-phase species and reactions, "gaas_b.gas" (Figure B.5). This file contains four species used in the deposition of Gallium Arsenide crystals: AsH3, GaMe3, CH4, and H2; the first and third have zero mole fractions in this problem.



```
ELEMENTS
Ga As H C
END
!
SPECIES
 AsH3
 GaMe3
 CH4
 H2
END
!
THERMO ALL
300. 1000. 3000.
| Default temperature ranges for thermo files:
300.   1000.   3000.
|
CH4                  121286C   1H   4            G  0300.00   5000.00  1000.00      1
 1.68347883E+00 1.02372356E-02-3.87512864E-06 6.78558487E-10-4.50342312E-14    2
-1.00807871E+04 9.62339497E+00 7.78741479E-01 1.74766835E-02-2.78340904E-05    3
 3.04970804E-08-1.22393068E-11-9.82522852E+03 1.37221947E+01                   4
H2                   121286H   2                 G  0300.00   5000.00  1000.00      1
 2.99142337E+00 7.00064411E-04-5.63382869E-08-9.23157818E-12 1.58275179E-15    2
-8.35033997E+02-1.35511017E+00 3.29812431E+00 8.24944174E-04-8.14301529E-07    3
-9.47543433E-11 4.13487224E-13-1.01252087E+03-3.29409409E+00                   4
AsH3                  92090As   1H   3    0     0G   300.000  3000.000 1000.00    0 1
 0.48852077E+01 0.38298892E-02-0.36824741E-06-0.38565741E-09 0.87018486E-13    2
 0.81938514E+04-0.28651702E+01 0.27935255E+01 0.83927420E-02 0.77386630E-06    3
-0.19897164E-08 0.79792984E-13 0.90261641E+04 0.89869089E+01                   4
GaMe3                 92090Ga   1C   3H   9     0G   300.000  3000.000 1000.00    0 1
 0.12968908E+02 0.15346084E-01-0.12010402E-05-0.18080676E-08 0.32430402E-12    2
-0.87401934E+04-0.36943215E+02 0.47942584E+01 0.30363396E-01-0.34483364E-06    3
-0.15581833E-07 0.46991998E-11-0.61177363E+04 0.71641846E+01                   4
END
```

*Figure B.5. Chemkin input file* `gaas_b.gas`*, which contains the four species and their thermodynamic data. No reactions are included.*

The solution of this problem requires only 2.16 seconds on an SGI workstation, 5 Newton iterations, and a total of 68 linear solve iterations. The solution across the domain at $y = 0.5$ is output using the `time_history_line` included function, as can be seen on the last lines of the input file. By plotting the output with "gnuplot," the temperature and mole fraction of GaMe3 across the width of the domain can analyzed, as in Figure B.6.

## B.3. Si3N4 Equilibrium

This example differs from the previous examples in that it is run on multiple processors, there are chemical reactions, and the steady-state solution is reached through time integration. The example uses a large gas-phase reaction mechanism for the formation of Silicon Nitride involving 17 species and 33 reactions. The species list and reaction mechanism are contained in the Chemkin input file "si3n4.gas," which is not shown here. An initial mixture of three reactants is set in a 2D domain at a high temperature and allowed to react until equilibrium. No spatial

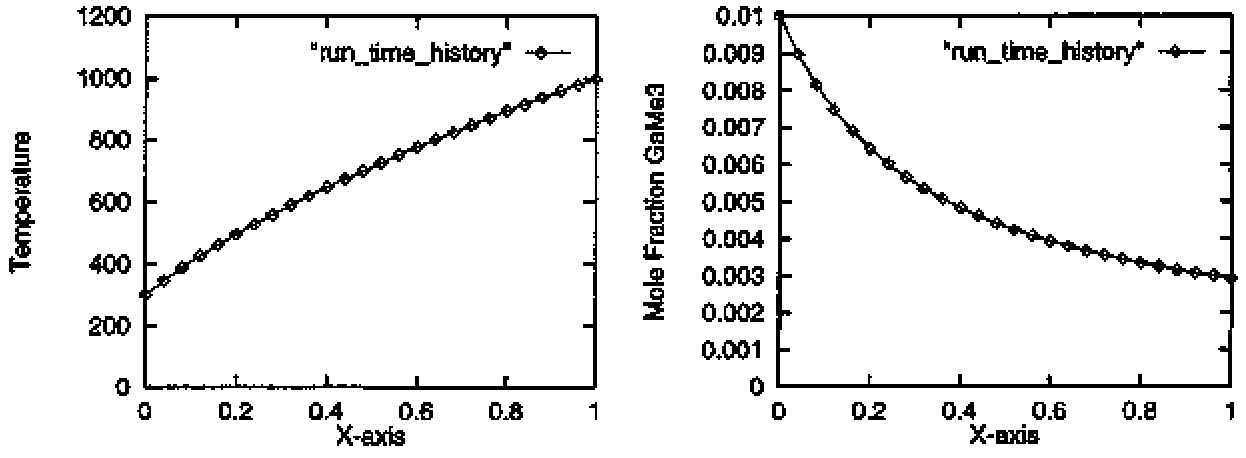


*Figure B.6. Profiles of temperature and GaMe3 mole fraction in the* Soret Effect *example problem. The temperature is fixed at both ends, and the mole fraction is fixed at the left side. The drop in the mole fraction as x increases is due solely to thermal diffusion.*

gradients are given in the problem, either as initial conditions or boundary conditions, so the solution is essentially 0D.

The input file for this problem can be seen in Figure B.7. An accurate transient solution of the problem is not desired; rather, only the solution at the final equilibrium state is of interest. Thus, the `pseudo` time integration option is used with a stopping point of 100 seconds. The use of only block-Jacobi scaling for preconditioning the matrix is adequate for many time-dependent problems, since the matrix is better conditioned than with the steady-state formulation.

The input file is set up for running on 8 processors, and requires that a load balance file "Meshes/testa-8-bKL.exoII" has been created. To run this problem in parallel on the Intel Paragon, the file "chem.bin" must first be created on this machine from the Chemkin input file by the following command:

```
> interp si3n4
```

To then solve the problem with MPSalsa, with an executable "salsa-smos" and the input file "input-si3n4," the user must type:

```
> yod -sz 8 salsa-smos input-si3n4
```

This run took 23 time steps to reach 100 seconds, and required 376 seconds.

Figure B.8 shows how the mole fractions of many species evolve with time. The data for these plots was output using the `time_history_points` function, which is called within function `user_out` and has data supplied to it at the bottom of the input file. The plots were made directly from this output using "gnuplot."

```
------------------------------------------------------------
              General Problem Specifications
------------------------------------------------------------
Problem type                        = mass_diff
Input FEM file                      = Meshes/tests exoII
LB file                             = Meshes/tests-$-DXL exoII
Output FEM file                     = run-out exoII
Number of processors                = 8
Cartesian or Cylindrical when 2D    = Cartesian
Interpolation Order                 = linear
Stabilization                       = default
Debug                               = 2
------------------------------------------------------------
                  Solution Specifications
------------------------------------------------------------
Solution Type                          = pseudo
Order of integration/continuation      = 1
Step Control                           = on
Relative Time Integration Error        = 4 0e-3
Initial Parameter Value                = 300 0
Initial Step Size                      = 1 0e-5
Maximum Number of Steps                = 75
Maximum Time or Parameter Value        = 100 0
------------------------------------------------------------
                  Solver Specifications
------------------------------------------------------------
Override Default Linearity Choice                = nonlinear

---------------- nonlinear solver subsection --------------

Number of Newton Iterations                      = 10
Use Modified Newton Iteration                    = no
Enable backtracking for residual reduction       = default
Choice for Inexact Newton Forcing Term           = 4
Calculate the Jacobian Numerically               = no
Solution Relative Error Tolerance                = 1 0e-3
Solution Absolute Error Tolerance                = 1 0e-8

---------------- linear solver subsection -----------------

Solution Algorithm                               = gmres
Convergence Norm                                 = 1
Preconditioner                                   = none
Polynomial                                       = LS,1
Scaling                                          = block_jacobi
Orthogonalization                                = classical
Size of Krylov subspace                          = 100
Maximum Linear Solve Iterations                  = 300
Linear Solver Normalized Residual Tolerance      = 1 0e-4
------------------------------------------------------------
                  Chemistry Specifications
------------------------------------------------------------
Energy  equation source terms           = on
Species equation source terms           = on
Pressure (atmospheres)                  = 1 0
Thermal Diffusion                       = off
Multicomponent Transport                = stefan_maxwell
Chemkin file                            = chem bin
Surface chemkin file                    = surf bin
Transport chemkin file                  = tran bin
------------------------------------------------------------
              Enclosure Radiation Specifications
------------------------------------------------------------
Enclosure Radiation source terms        = off
------------------------------------------------------------
          Material ID Specifications
------------------------------------------------------------
Number of Materials                 = 1
CHEMKIN                             = 0    "Graphite"
         ELEM_BLOCK_IDS             = 1       2

         XMF_0       N2       0 5
         XMF_0      NH3       0 3
         XMF_0     SIF4       0 2

         T_INIT                     = 1700
END Material ID Specifications
```

```
------------------------------------------------------------
        Boundary Condition Specifications
------------------------------------------------------------
Number of Generalized Surfaces = 0
Number of BC                   = 0
#
------------------------------------------------------------
        Initial Guess/Condition Specifications
------------------------------------------------------------
Set Initial Condition/Guess                 = constant 0.0
Apply function                              = no
Time Index to Restart From                  = 1
------------------------------------------------------------
                   Output Specifications
------------------------------------------------------------
User Defined Output                                   = yes
Parallel Output                                       = no
Scalar Output                                         = no
Time Index to Output To                               = 2
Nodal variable output times
        every 2 steps

Number of nodal output variables                      = 1
Nodal variable names
        Mass_fraction

Number of global output variables                     = 2
Global variable names
        Delta_time
        Time_index

Test Exact Solution Flag                              = 0
Name of Exact Solution Function                       = f_xx_yy
------------------------------------------------------------
                 Parallel I/O section
------------------------------------------------------------
Machine                                     = paragon
Staged writes                               = yes

----------------
ncube subsection
----------------

Number of controllers                       = 8
Disks per controller                        = 1
Root location                               = //df
Subdirectory                                = jns/tests
Offset numbering from zero                  = 0

------------------
paragon subsection
------------------

Number of RAID controllers                  = 8
Root location                               = /pfs/io_
Subdirectory                                = tmp/ags
Offset numbering from zero                  = 1
------------------------------------------------------------
      Data Specification for User's Functions
------------------------------------------------------------
Number of functions to pass data to    =  1

Function Name    =  time_history_points 1
$
$N_DATA = TABLE  1  2
    11   11
$
```

*Figure B.7. Input file for the* Si3N4 Equilibrium *example problem.*

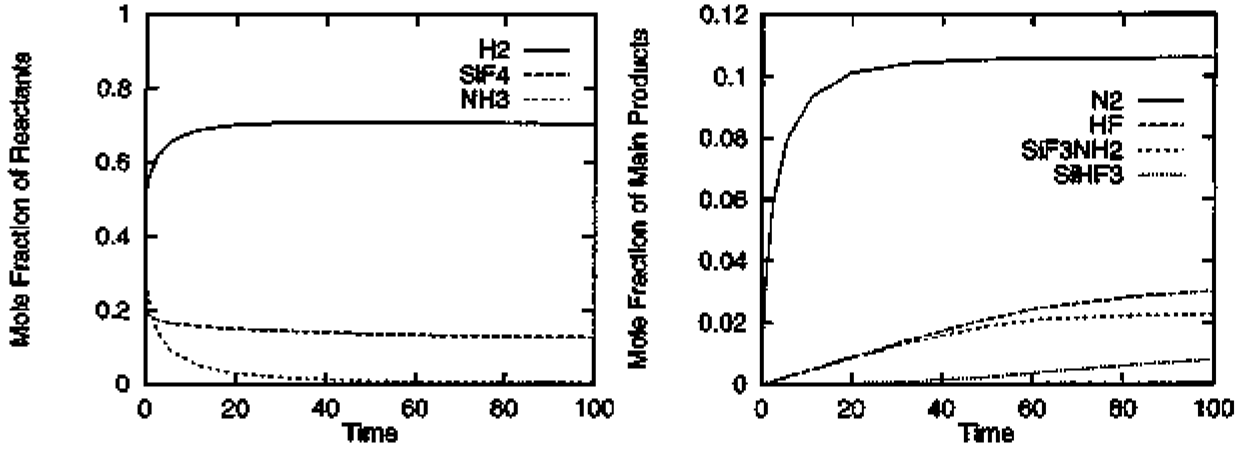


*Figure B.8. Evolution in time of mole fractions of major species in the* Si3N4 Equilibrium *problem. The first plot shows the three reactants, while the second shows the major products of the reactions. Since the pseudo time integration scheme was used, these histories are not time accurate.*

## B.4. Surface Reaction

This simple reaction-diffusion problem illustrates the use of `surface_chemkin_bc`, the function used to impose surface reactions as boundary conditions by interfacing with the Surface Chemkin library (see Appendix A.1.1). Just as Chemkin is used for information on gas-phase species, reactions, and properties, Surface Chemkin is used to access this information about the surface and underlying bulk solid.

The problem is defined in a 2D box and uses the mechanism for the deposition of Gallium Arsenide semiconductor crystals. This mechanism contains 17 gas-phase species, 24 gas-phase reactions, 6 surface species, 38 surface reactions, and 2 bulk species. The surface reactions occur on the left side of the box, and Dirichlet conditions for the main reactants and carrier gas are set on the right side, as shown in Figure B.9. The system is assumed isothermal (at 913K); no-slip velocities are imposed on all walls and no penetration is assumed on the top and bottom. At the reacting surface, the normal velocity is not zero, but is set equal to the total mass flux per unit area into the surface, divided by the density. This term is often called the Stefan velocity (see equation (A.3)). At the right side, the normal momentum balance has a natural condition applied that sets the normal component of the normal stress to zero. This boundary condition allows for a non-zero velocity at this surface.

The surface site fractions of surface species and the bulk fractions are also unknowns in this problem. To specify their values, we use a quasi-steady state assumption that these species are always in equilibrium with the gas phase. This approximation adds no error for a steady-state

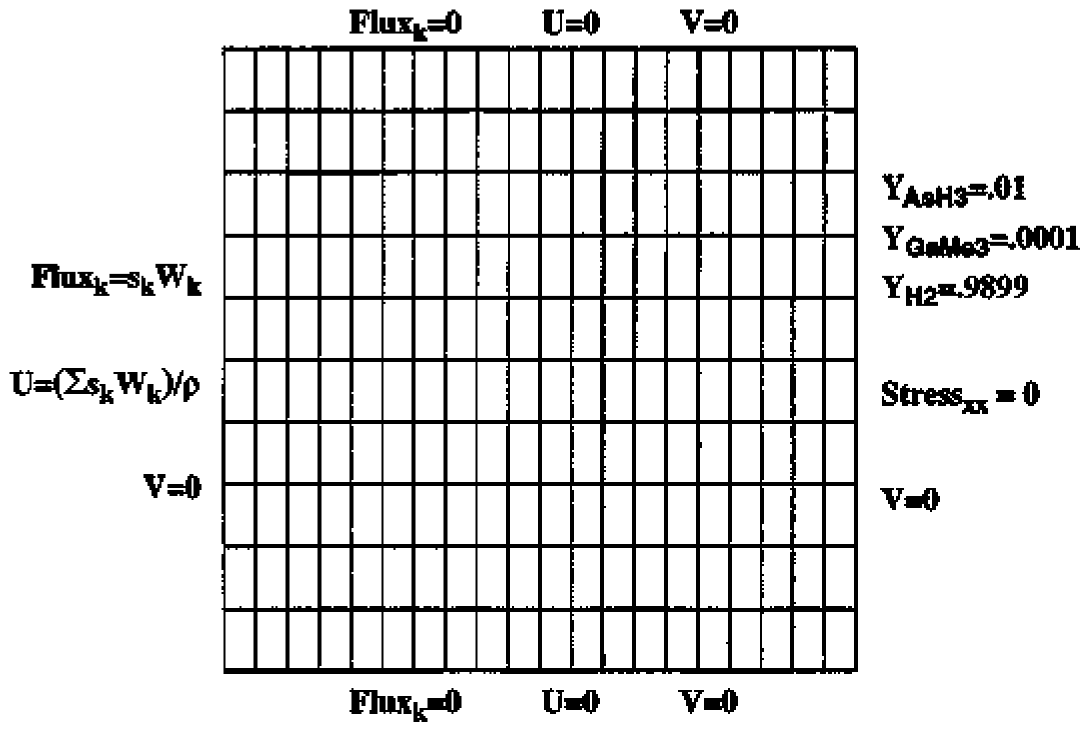


*Figure B.9. 200-element mesh and boundary conditions for the* Surface Reaction *example problem.* $s_k$ *is the molar production rate of species k due to the surface reaction,* $W_k$ *is the molecular weight os species k, and* $\rho$ *is the density. The nonzero velocity due to surface reaction is called the Stefan velocity.*

solution and is a good approximation in transient problems because of the relative speed of surface reactions. Using the requirement that the generation rate of any surface species is equal to its consumption rate, and given the gas-phase species mole fractions, these unknowns can be solved for implicitly and removed from the problem.

The input file for this problem is shown in Figure B.10. There are 20 unknowns per node in this problem: 2 velocities, 1 pressure, and 17 species. The steady solution is solved for directly using a preconditioned GMRES method, starting from an initial guess where 3 species have nonzero mole fractions (see the `XMF_0` lines in the Materials Specifications section). The `surface_chemkin_bc` boundary condition function is used for reacting surfaces. The Stefan velocity is set as a dependent Dirichlet condition where the value comes from the `surface_chemkin_bc` function. (The DEPENDENT keyword in this boundary condition specifies that Jacobian entries are included for this term.) The same function is used for the species balance equations, though in this case it is a Neumann boundary condition since it is a specification on the flux.

There is an option with the `surface_chemkin_bc` to input initial guesses for the surface site and bulk fractions. Since the equations for these species can be highly nonlinear, there

```
-------------------------------------------------------------
              General Problem Specifications
-------------------------------------------------------------
Problem type                       = fluid_flow_mass
Input FEM file                     = Meshes/box200 exoII
LB file                            = BKL exoII
Output FEM file                    = run_out exoII
Number of processors               = 1
Cartesian or Cylindrical when 2D   = Cartesian
Interpolation Order                = linear
Stabilization                      = default
Debug                              = 2
-------------------------------------------------------------
                 Solution Specifications
-------------------------------------------------------------
Solution Type                           = steady
Order of integration/continuation       = 1
Step Control                            = on
Relative Time Integration Error         = 5 0e-3
Initial Parameter Value                 = 300 0
Initial Step Size                       = 1 0e-7
Maximum Number of Steps                 = 4
Maximum Time or Parameter Value         = 10
-------------------------------------------------------------
                  Solver Specifications
-------------------------------------------------------------
Override Default Linearity Choice                 = default

-- nonlinear solver subsection

Number of Newton Iterations                       = 12
Use Modified Newton Iteration                     = no
Enable backtracking for residual reduction        = no
Choice for Inexact Newton Forcing Term            = 4
Calculate the Jacobian Numerically                = no
Solution Relative Error Tolerance                 = 1 0e-3
Solution Absolute Error Tolerance                 = 1 0e-8

-- linear solver subsection

Solution Algorithm                                = gmres
Convergence Norm                                  = 0
Preconditioner                                    = no_overlap_ilu
Polynomial                                        = LS,1
Scaling                                           = row_sum
Orthogonalization                                 = classical
Size of Krylov subspace                           = 50
Maximum Linear Solve Iterations                   = 100
Linear Solver Normalized Residual Tolerance       = 3 0e-3
-------------------------------------------------------------
                 Chemistry Specifications
-------------------------------------------------------------
Energy  equation source terms       = on
Species equation source terms       = on
Pressure (atmospheres)              = 0 1
Thermal Diffusion                   = off
Multicomponent Transport            = stefan_maxwell
Chemkin file                        = chem bin
Surface chemkin file                = surf bin
Transport chemkin file              = tran bin
-------------------------------------------------------------
           Enclosure Radiation Specifications
-------------------------------------------------------------
Enclosure Radiation source terms    = off
-------------------------------------------------------------
        Material ID Specifications
-------------------------------------------------------------
Number of Materials                 = 1
CHEMKIN                             = 0    "gas"
         ELEM_BLOCK_IDS             = 1

# T_INIT set the Temperature for this isothermal problem
         T_INIT          = 913 0

         U_INIT        = 0 0
         V_INIT        = 0 0
         P_INIT         = 0 0
         XMF_0   AsH3       0 001
         XMF_0 GaMe3        0 0001
         XMF_0     H2       0 9989

END Material ID Specifications
```

```
-------------------------------------------------------------
        Boundary Condition Specifications
-------------------------------------------------------------
Number of Generalized Surfaces = 0

Number of BC                    = 9
#
BC    = U_BC DIRICHLET SS 4 DEPENDENT surface_chemkin_bc 0
BC    = U_BC DIRICHLET SS 1 INDEPENDENT 0   0
BC    = U_BC DIRICHLET SS 3 INDEPENDENT 0   0
#
BC    = V_BC DIRICHLET SS 1 INDEPENDENT 0   0
BC    = V_BC DIRICHLET SS 2 INDEPENDENT 0   0
BC    = V_BC DIRICHLET SS 3 INDEPENDENT 0   0
BC    = V_BC DIRICHLET SS 4 INDEPENDENT 0   0
#
BC    = Y_BC DIRICHLET SS 2 INDEPENDENT f_mole_fraction 1
        SPECIES_LIST = ALL
       BC_DATA =  001  0  0  0  0  0  0  0  0  0001  0  0  0  0
 0  0  9989
BC    = Y_BC NEUMANN    SS 4 DEPENDENT surface_chemkin_bc 2
        SPECIES_LIST = ALL
        SURF_SPECIES_LIST = GaMe(S) Ga(S) GaH(S) AsH(S) AsMe(S)
As(S)
        BC_DATA = FLOAT 1 0e-5 0 5 1 0e-5 1 0e-5 1 0e-5 0 5
        SURF_SPECIES_LIST = Ga-GaAs(D) As-GaAs(D)
        BC_DATA = FLOAT 1 0 1 0
-------------------------------------------------------------
        Initial Guess/Condition Specifications
-------------------------------------------------------------
Set Initial Condition/Guess                   = constant 0 0
Apply function                                = no
Time Index to Restart From                    = 1
-------------------------------------------------------------
                  Output Specifications
-------------------------------------------------------------
User Defined Output                                  = yes
Parallel Output                                      = no
Scalar Output                                        = yes
Time Index to Output To                              = 1
Nodal variable output times
        every 1 steps

Number of nodal output variables                     = 3
Nodal variable names
        Velocity
        Pressure
        Mass_Fraction

Number of global output variables                    = 0
Global variable names

Test Exact Solution Flag                             = 0
Name of Exact Solution Function                      = f_xx_yy
-------------------------------------------------------------
                  Parallel I/O section
-------------------------------------------------------------
Machine                                       = paragon
Staged writes                                 = yes

----------------------
paragon subsection
----------------------

Number of RAID controllers                    = 8
Root location                                 = /pfs/io_
Subdirectory                                  = tmp/kdd/t13
Offset numbering from zero                    = 23
-------------------------------------------------------------
      Data Specification for User's Functions
-------------------------------------------------------------
Number of functions to pass data to    =  0
```

*Figure B.10.  Input file for the* Surface Reaction *example problem.*

are initial guesses that do not lead to a converged solution, and sometimes there are multiple solutions. The initial guesses are input using the SURF_SPECIES_LIST keyword, as can be seen in the input file. The default initial guess is equal fractions of all species within a given surface or bulk phase. For the mechanism in this example, the surface reaction calculations fail with the default initial guess. The initial guess is used only the first time the surface reaction calculations are computed; for subsequent Newton iterations and time steps, the previous calculation of surface site and bulk fractions are used as the initial guess.

The steady-state solution for the 4620 unknowns in this problem required 4 Newton iterations and 89 seconds on an SGI workstation. A visualization of the solution is presented in Figure B.11. The weak flow driven by the Stefan velocity is shown with velocity vectors, as are contours of one of the species generated by the surface reactions and consumed in gas-phase reactions. The vertical contours show that the flow is too weak for convection to distort the 1D diffusion-reaction problem.

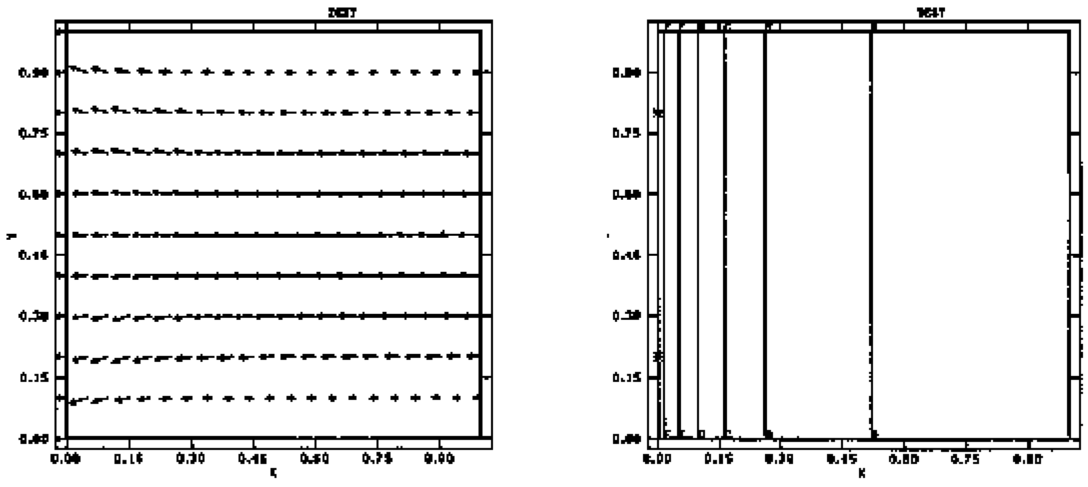


*Figure B.11. Visualization of the solution for the* Surface Reaction *example problem. The deposition on the left wall drives a velocity to the left, as shown in the plot on the left. The velocity is nearly uniform near the wall, but is more parabolic at the source on the right side. Shown on the right are mole fraction contours of the* H *atom, which is produced at the surface.*

## Appendix C. Fluid Mechanics and Heat Transfer Examples

The example problems in Section C.3 through Section C.5 were developed, run, and written up by Professor Michael Jensen of the Mechanical Engineering Department of Rensselaer Polytechnic Institute during a sabbatical at Sandia National Laboratories in Spring 1996. Exhaustive mesh independence studies were not done for any of the examples in Section C.3 through Section C.5, but the meshes were refined to adequately show agreement with data from the literature. For these examples, the mks unit system was used; that is, the units used on all the quantities are length (m); velocity (m/s); temperature (K); pressure (N/m^2); heat flux (W/m^2); density (kg/m^3); specific heat (J/kgK); thermal conductivity (W/mK); and dynamic viscosity (Ns/m^2).

### C.1. Navier-Stokes 3D Exact Solution

An analytic solution to the Navier-Stokes equations for a three-dimensional time-dependent problem is known for a generalized Beltrami-type flow [11]. We use this problem to demonstrate the solution of a transient fluid mechanics system and to document the convergence properties of our implementation of the finite element method.

In MPSalsa, the function `f_3d_navier_stokes` provides the exact solution for this flow in a cube of unit length when these same functions, evaluated at all boundaries, are imposed as boundary conditions:

$$\begin{aligned}
u &= -ae^{-d^2t}\left(e^{ax}\sin(ay+dz) + e^{az}\cos(ax+dy)\right) \\
v &= -ae^{-d^2t}\left(e^{ay}\sin(az+dx) + e^{ax}\cos(ax+dz)\right) \\
w &= -ae^{-d^2t}\left(e^{az}\sin(ax+dy) + e^{ay}\cos(az+dx)\right) \\
p &= -\frac{1}{2}a^2e^{-2d^2t}\Big(e^{2ax} + e^{2ay} + e^{2az} + \\
&\qquad 2\sin(ax+dy)\cos(az+dx)\,e^{a(y+z)} + \\
&\qquad 2\sin(ay+dz)\cos(ax+dy)\,e^{a(x+z)} + \\
&\qquad 2\sin(az+dx)\cos(ay+dz)\,e^{a(x+y)}\Big) \\
a &= 0.25\pi \\
d &= 0.5\pi
\end{aligned} \tag{C.1}$$

The MPSalsa input file for this test problem is shown in Figure C.1. The first line specifies that a fluid mechanics problem is to be solved. A linear spatial approximation is to be used. A time-accurate transient solution method with a second-order time integration scheme and variable time step is selected. The run is set to terminate at a time of 0.1 seconds. As can be seen in the Boundary Condition Specifications section, Dirichlet boundary conditions computed by the function `f_3d_navier_stokes` are prescribed for all velocity and pressure unknowns on all domain boundaries. This same function used to specify the initial conditions. In addition, the exact solution is compared with the computed solution for convergence analyses by setting `f_3d_navier_stokes` in the input file as the exact solution.

The input ExodusII mesh is an 8x8x8-element mesh with 729 nodes and 2916 total unknowns. The same problem was solved using discretizations of 4x4x4, 16x16x16, and 32x32x32 elements. Details of the four runs are show in Table C.1. All runs required 27 time steps to reach 0.1 seconds.

| Number of elements in 1D | Total Number of Elements | # of Intel Paragon Processors | CPU seconds | $L^2$-error of Velocity in the $x$-direction at 0.1 sec. | $L^2$ -error of Pressure at 0.1 sec |
|---|---|---|---|---|---|
| 4 | 64 | 1 | 305 | 1.008e-03 | 1.904e-02 |
| 8 | 512 | 16 | 308 | 2.781e-04 | 1.183e-02 |
| 16 | 4096 | 64 | 452 | 6.512e-05 | 1.643e-03 |
| 32 | 32,768 | 128 | 1543 | 1.381e-05 | 5.090e-04 |

*Table C.1. Details of the mesh convergence calculations for the* Navier-Stokes 3D Exact Solution *problem.*

The error in the computed solution as compared to the exact solution is presented in Table C.1 and shown graphically in Figure C.2. The $L^2$-norms of the error in the $x$-component of the velocity and in the pressure unknown are plotted versus the element size. The slopes of the lines connecting the results for the coarsest mesh and the finest mesh on the log-log plot are near 2, the expected value for the linear discretization scheme.

## C.2. Lid-Driven Cavity Problem

The lid-driven cavity problem is a two-dimensional fluid mechanics problem on a square domain that has often been used as a benchmark problem [19]. The fluid is confined in the square, but the top surface is pulled horizontally, driving clockwise flow. The geometry, boundary conditions, and 64x64-element mesh are shown in Figure C.3.

The input file for this example is shown in Figure C.4. The viscosity and density are set to one, so that the velocity is equal to the Reynolds number. This problem is increasingly difficult to solve as the Reynolds number is increased. SUPG stabilization is turned on (in the General

```
------------------------------------------------------------
              General Problem Specifications
------------------------------------------------------------
Problem type                          = fluid_flow
Input FEM file                        =   /Meshes/box_3d_8 exoII
LB file                               =   /Meshes/box_3d_8-m16-
b8L nemI
Output FEM file                       = box_3d_out exoII
Number of processors                  = 16
Cartesian or Cylindrical when 2D = Cartesian
Interpolation Order                   = linear
Stabilization                         = default
Debug                                 = 1
------------------------------------------------------------
                 Solution Specifications
------------------------------------------------------------
Solution Type                           = transient
Order of integration/continuation       = 2
Step Control                            = on
Relative Time Integration Error         = 1 0e-5
Initial Parameter Value                 = 10 0
Initial Step Size                       = 1 0e-5
Maximum Number of Steps                 = 2000
Maximum Time or Parameter Value         = 0 1
------------------------------------------------------------
                  Solver Specifications
------------------------------------------------------------
Override Default Linearity Choice                 = default

---------------- nonlinear solver subsection  --------------

Number of Newton Iterations                       = 25
Use Modified Newton Iteration                     = no
Enable backtracking for residual reduction        = default
Choice for Inexact Newton Forcing Term            = 0
Calculate the Jacobian Numerically                = no
Solution Relative Error Tolerance                 = 1 0e-6
Solution Absolute Error Tolerance                 = 1 0e-8

---------------- linear solver subsection  -----------------

Solution Algorithm                                = gmres
Convergence Norm                                  = 0
Preconditioner                                    = none
Polynomial                                        = LS.1
Scaling                                           = block_jacobi
Orthogonalization                                 = classical
Size of Krylov subspace                           = 64
Maximum Linear Solve Iterations                   = 200
Linear Solver Normalized Residual Tolerance       = 1 0e-8
------------------------------------------------------------
                 Chemistry Specifications
------------------------------------------------------------
Energy  equation source terms           = off
Species equation source terms           = off
Pressure (atmospheres)                  = 0 09210526
Thermal Diffusion                       = on
Multicomponent Transport                = stefan_maxwell
Chemkin file                            = chem bin
Surface chemkin file                    = surf bin
Transport chemkin file                  = tran bin
------------------------------------------------------------
           Enclosure Radiation Specifications
------------------------------------------------------------
Enclosure Radiation source terms        = off

------------------------------------------------------------
     Material ID Specifications
------------------------------------------------------------
Number of Materials                   = 1
NEWTONIAN                             = 0    "Air"
         ELEM_BLOCK_IDS               = 1
      VISCOSITY        = 1 0
      DENSITY          = 1 0
END Material ID Specifications
```

```
------------------------------------------------------------
     Boundary Condition Specifications
------------------------------------------------------------
Number of Generalized Surfaces    = 0
Number of BC                      = 24
# Prescribed Dirichlet conditions on all boundaries
BC = U_BC DIRICHLET SS 1 INDEPENDENT f_3d_navier_stokes 0
BC = V_BC DIRICHLET SS 1 INDEPENDENT f_3d_navier_stokes 0
BC = W_BC DIRICHLET SS 1 INDEPENDENT f_3d_navier_stokes 0
BC = P_BC DIRICHLET SS 1 INDEPENDENT f_3d_navier_stokes 0
#
BC = U_BC DIRICHLET SS 2 INDEPENDENT f_3d_navier_stokes 0
BC = V_BC DIRICHLET SS 2 INDEPENDENT f_3d_navier_stokes 0
BC = W_BC DIRICHLET SS 2 INDEPENDENT f_3d_navier_stokes 0
BC = P_BC DIRICHLET SS 2 INDEPENDENT f_3d_navier_stokes 0
#
BC = U_BC DIRICHLET SS 3 INDEPENDENT f_3d_navier_stokes 0
BC = V_BC DIRICHLET SS 3 INDEPENDENT f_3d_navier_stokes 0
BC = W_BC DIRICHLET SS 3 INDEPENDENT f_3d_navier_stokes 0
BC = P_BC DIRICHLET SS 3 INDEPENDENT f_3d_navier_stokes 0
#
BC = U_BC DIRICHLET SS 4 INDEPENDENT f_3d_navier_stokes 0
BC = V_BC DIRICHLET SS 4 INDEPENDENT f_3d_navier_stokes 0
BC = W_BC DIRICHLET SS 4 INDEPENDENT f_3d_navier_stokes 0
BC = P_BC DIRICHLET SS 4 INDEPENDENT f_3d_navier_stokes 0
#
BC = U_BC DIRICHLET SS 5 INDEPENDENT f_3d_navier_stokes 0
BC = V_BC DIRICHLET SS 5 INDEPENDENT f_3d_navier_stokes 0
BC = W_BC DIRICHLET SS 5 INDEPENDENT f_3d_navier_stokes 0
BC = P_BC DIRICHLET SS 5 INDEPENDENT f_3d_navier_stokes 0
#
BC = U_BC DIRICHLET SS 6 INDEPENDENT f_3d_navier_stokes 0
BC = V_BC DIRICHLET SS 6 INDEPENDENT f_3d_navier_stokes 0
BC = W_BC DIRICHLET SS 6 INDEPENDENT f_3d_navier_stokes 0
BC = P_BC DIRICHLET SS 6 INDEPENDENT f_3d_navier_stokes 0
------------------------------------------------------------
        Initial Guess/Condition Specifications
------------------------------------------------------------
Set Initial Condition/Guess                 = constant 0 0
Apply function                              = f_3d_navier_stokes
Time Index to Restart From                  = 0
------------------------------------------------------------
                 Output Specifications
------------------------------------------------------------
User Defined Output                                  = yes
Parallel Output                                      = no
Scalar Output                                        = no
Time Index to Output To                              = 0
Nodal variable output times
        every 3 steps

Number of nodal output variables                     = 2
Nodal variable names
        Velocity
        Pressure

Number of global output variables                    = 0
Global variable names

Test Exact Solution Flag                             = 1 SUMMARY
Name of Exact Solution Function = f_3d_navier_stokes
------------------------------------------------------------
                Parallel I/O section
------------------------------------------------------------
Machine                                     = paragon
Staged writes                               = yes
----------------
ncube subsection
----------------
Number of controllers                       = 8
Disks per controller                        = 1
Root location                               = //df
Subdirectory                                = jns/tests
Offset numbering from zero                  = 0
------------------
paragon subsection
------------------
Number of RAID controllers                  = 26
Root location                               = /pfs/io_
Subdirectory                                = tmp/ags/ti43
Offset numbering from zero                  = 23
------------------------------------------------------------
      Data Specification for User's Functions
------------------------------------------------------------
Number of functions to pass data to    =  0
```

*Figure C.1. Input file for the* Navier-Stokes 3D Exact Solution *example problem.*

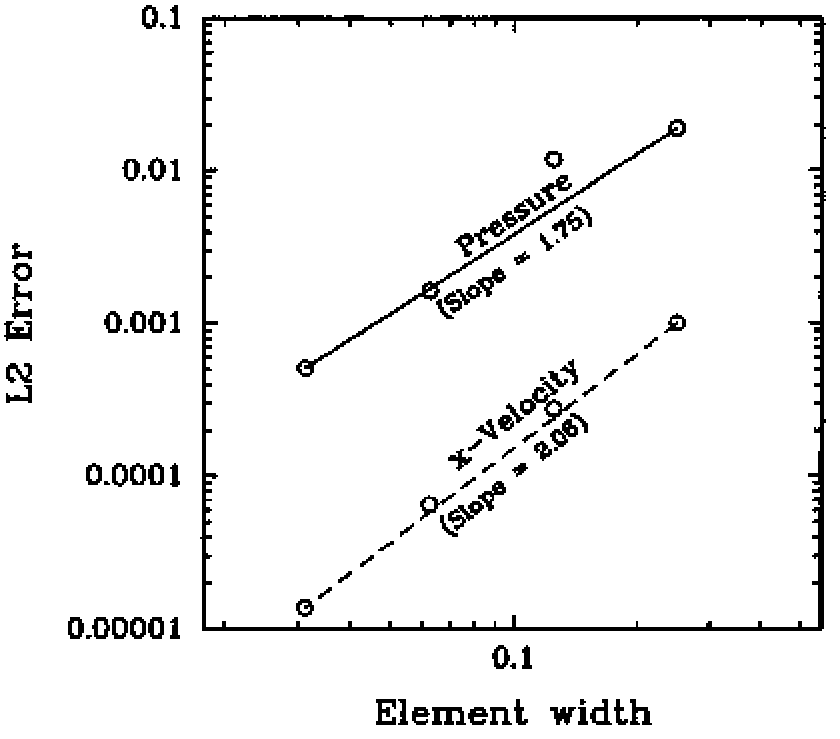


*Figure C.2. Log-log plot of the $L^2$-error in the solution versus the element width for the* Navier-Stokes 3D Exact Solution *problem. Second-order convergence with respect to the mesh spacing is observed.*

Problem Specifications section of the input file), which reduces the oscillations in highly-convective flows and greatly improves convergence.

The backtracking algorithm in the nonlinear solver is also turned on. For this calculation, which starts from a trivial initial guess and attempts to reach a steady state at a Reynolds number of 1500, Newton's method without backtracking diverges. With backtracking, this calculation converged to a steady state in 11 Newton iterations, which took 229 seconds on 16 processors of the Intel Paragon.

In Section 5.3, this example problem was used to demonstrate the method of mesh sequencing for obtaining a converged solution to a difficult problem. For large problems that are spread across many processors, the ILU (domain decomposition) preconditioners are not as robust. In many cases, the same problem on a coarser mesh and spread across fewer processors will converge more readily. Mesh sequencing is a method to capitalize on this phenomena by first solving the problem on a coarse mesh, interpolating the converged solution to a finer mesh, and then using this solution as an initial guess on the fine, accurate mesh. See Table 5.2 in Section 5.3 for an example of the benefit of this approach.

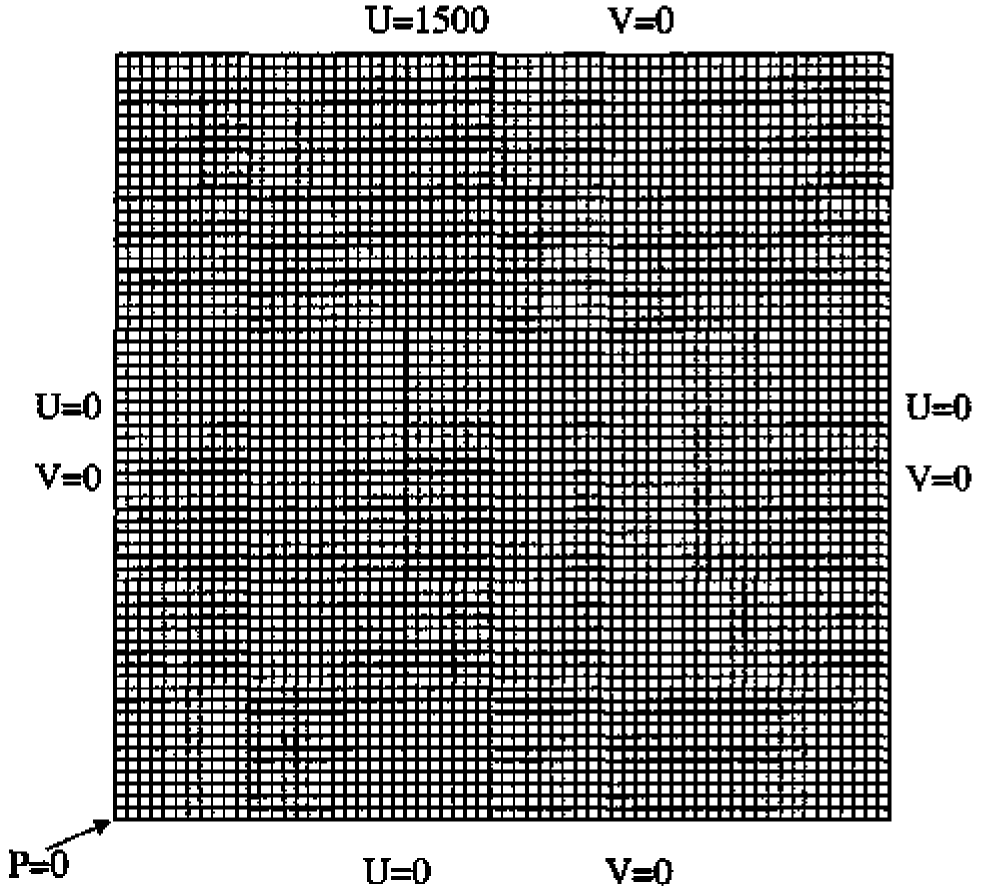


*Figure C.3. 4096-element mesh and boundary conditions for the* Lid-Driven Cavity *example problem.*

## C.3. Hydrodynamically Developing Flow in an Infinite Parallel Plate Channel

Developing steady laminar flow in the entrance region of a straight parallel plate channel is demonstrated in this example. To resolve the flow near the inlet, a mesh that was finer near the inlet than at the outlet was used. The mesh was also refined near the lower wall boundary. The entire mesh had 500x60 elements. A small section of the domain in the entrance region is shown in Figure C.5 to show the expanding mesh. Advantage is taken of the line of symmetry through the channel centerline. An expanding grid is used from the wall to the centerline and from the entrance along the channel. The upper plate is located 0.5 units from the channel centerline, and the channel has a length of 10. The upper plate is designated Side Set 1; the outflow boundary is Side Set 2; the channel centerline is Side Set 3; and the inlet boundary is Side Set 4.

```
-------------------------------------------------------------
               General Problem Specifications
-------------------------------------------------------------
Problem type                           = fluid_flow
Input FEM file                         = Meshes/box_0064 exoII
LB file                               = Meshes/box_0064-m16-bKL nemI
Output FEM file                        = run_out.exoII
Number of processors                   = 16
Cartesian or Cylindrical when 2D       = Cartesian
Interpolation Order                    = linear
Stabilization                          = supg
Debug                                  = 2
-------------------------------------------------------------
                  Solution Specifications
-------------------------------------------------------------
Solution Type                             = steady
Order of integration/continuation         = 1
Step Control                              = on
Relative Time Integration Error           = 1.0e-3
Initial Parameter Value                   = 10.0
Initial Step Size                         = 1.0e-2
Maximum Number of Steps                   = 80
Maximum Time or Parameter Value           = 1.0e+2
-------------------------------------------------------------
                   Solver Specifications
-------------------------------------------------------------
Override Default Linearity Choice                    = default

---------------- nonlinear solver subsection: --------------

Number of Newton Iterations                          = 50
Use Modified Newton Iteration                        = no
Enable backtracking for residual reduction           = yes
Choice for Inexact Newton Forcing Term               = 4
Calculate the Jacobian Numerically                   = no
Solution Relative Error Tolerance                    = 1.0e-2
Solution Absolute Error Tolerance                    = 1.0e-5

---------------- linear solver subsection: ----------------

Solution Algorithm                                   = gmres
Convergence Norm                                     = 0
Preconditioner                                       = no_overlap_ilu
Polynomial                                           = LS,1
Scaling                                              = row_sum
Orthogonalization                                    = classical
Size of Krylov subspace                              = 200
Maximum Linear Solve Iterations                      = 500
Linear Solver Normalized Residual Tolerance          = 1.0e-4
-------------------------------------------------------------
                  Chemistry Specifications
-------------------------------------------------------------
Energy  equation source terms          = off
Species equation source terms          = off
Pressure (atmospheres)                 = 0 09210526
Thermal Diffusion                      = on
Multicomponent Transport               = stefan_maxwell
Chemkin file                           = chem.bin
Surface chemkin file                   = surf.bin
Transport chemkin file                 = tran.bin
-------------------------------------------------------------
             Enclosure Radiation Specifications
-------------------------------------------------------------
Enclosure Radiation source terms       = off

-------------------------------------------------------------
      Material ID Specifications
-------------------------------------------------------------
Number of Materials                    = 1
$SOLID                                 = 0    "Air"
         ELEM_BLOCK_IDS                = 1
      VISCOSITY         = 1 0
      DENSITY           = 1.0
END Material ID Specifications
```

```
-------------------------------------------------------------
      Boundary Condition Specifications
-------------------------------------------------------------
Number of Generalized Surfaces     = 0
Number of BC                       = 9
# Upper moving wall
BC = U_BC DIRICHLET NS 3 INDEPENDENT 1500 0 0
BC = V_BC DIRICHLET NS 3 INDEPENDENT 0 0 0

# No slip boundary conditions on all surfaces
BC = U_BC DIRICHLET NS 1 INDEPENDENT 0 0 0
BC = V_BC DIRICHLET NS 1 INDEPENDENT 0.0 0
BC = U_BC DIRICHLET NS 2 INDEPENDENT 0 0 0
BC = V_BC DIRICHLET NS 2 INDEPENDENT 0.0 0
BC = U_BC DIRICHLET NS 4 INDEPENDENT 0.0 0
BC = V_BC DIRICHLET NS 4 INDEPENDENT 0 0 0

#    PRESSURE DATUM SET AT A SINGLE NODE FOR PROBLEM WITH
#    NO NATURAL OR SPECIFIED STRESS BOUNDARY
BC = P_BC DIRICHLET NS 5 INDEPENDENT 0 0 0
-------------------------------------------------------------
        Initial Guess/Condition Specifications
-------------------------------------------------------------
Set Initial Condition/Guess                   = constant 0 0
Apply function                                = no
Time Index to Restart From                    = 1
-------------------------------------------------------------
                  Output Specifications
-------------------------------------------------------------
User Defined Output                                  = yes
Parallel Output                                      = no
Scalar Output                                        = yes
Time Index to Output To                              = 1
Nodal variable output times
        every 2 steps

Number of nodal output variables                     = 2
Nodal variable names:
        Velocity
        Pressure

Number of global output variables                    = 0
Global variable names

Test Exact Solution Flag                             = 0
Name of Exact Solution Function                      = f_xx_yy
-------------------------------------------------------------
                 Parallel I/O section
-------------------------------------------------------------
Machine                                       = paragon
Staged writes                                 = yes
----------------
nCube subsection
----------------
Number of controllers                         = 8
Disks per controller                          = 1
Root location                                 = //df
Subdirectory                                  = jns/tests
Offset numbering from zero                    = 0
------------------
paragon subsection
------------------
Number of RAID controllers                    = 26
Root location                                 = /pfs/io_
Subdirectory                                  = tmp/mgs/tid3
Offset numbering from zero                    = 23
-------------------------------------------------------------
      Data Specification for User's Functions
-------------------------------------------------------------
Number of functions to pass data to    =  0
```

*Figure C.4. Input file for the* Lid-Driven Cavity *example problem.*

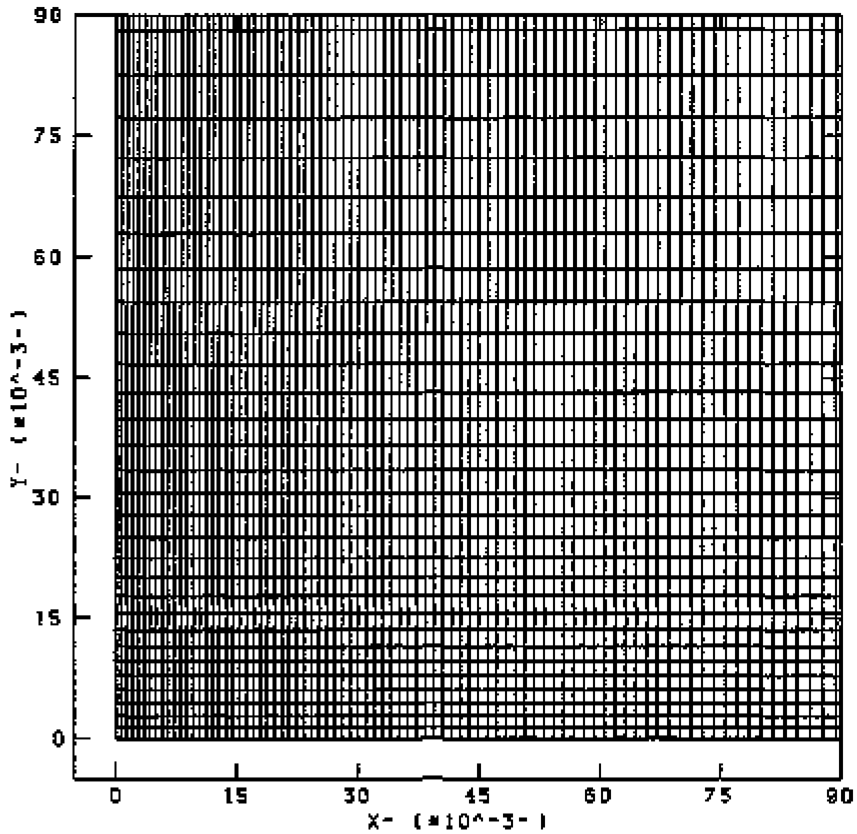


*Figure C.5. Expanding mesh of the entrance region for developing flow between parallel plates.*

A uniform velocity profile is provided at the entrance to the channel. No slip is imposed at the solid wall, and no shear is set at both the channel centerline and the outflow boundary; transverse velocities are set to zero on all side sets. The MPSalsa input file is listed in Figure C.6.

Shown in Figure C.7 is the developing velocity profile along the channel; comparison is made against results from a similar calculation using the finite difference algorithm SIMPLER [36] on a coarser grid. (The characteristic overshoot in velocity at locations near the entrance is physically possible and can be obtained numerically using the appropriate entrance and boundary conditions, as discussed in Shah and London [45].) The analytic solution for fully-developed flow in a channel predicts that the product of the friction factor and the Reynolds number is 24.0. The value of 23.97 calculated by MPSalsa at the exit of the channel compares well with the analytic result.

## C.4. Thermally Developing Flow in an Infinite Parallel Plate Channel

A variation of the example in Appendix C.3 is to impose a hydrodynamically fully-developed flow (parabolic velocity profile) at the entrance of the channel and to heat the wall at a constant heat flux. The mesh used in Appendix C.3 is also used for this example (Figure C.5). The

```
-----------------------------------------------------------------
                 General Problem Specifications
-----------------------------------------------------------------
Problem type                          = fluid_flow
Input FEM file                        = rect exoII
LB file                               = rect-32-BKL.exoII
Output FEM file                       = rectFM-out exoII
Number of processors                  = 32
Cartesian or Cylindrical when 2D      = Cartesian
Interpolation Order                   = linear
Stabilization                         = supg
Debug                                 = 2
-----------------------------------------------------------------
                    Solution Specifications
-----------------------------------------------------------------
Solution Type                            = steady
Order of integration/continuation        = 1
Step Control                             = on
Relative Time Integration Error          = 1 0e-3
Initial Parameter Value                  = 10 0
Initial Step Size                        = 30 0
Maximum Number of Steps                  = 6
Maximum Time or Parameter Value          = 1 0e+2
-----------------------------------------------------------------
                     Solver Specifications
-----------------------------------------------------------------
Override Default Linearity Choice                   = default

---------------- nonlinear solver subsection ----------------

Number of Newton Iterations                         = 60
Use Modified Newton Iteration                       = no
Enable backtracking for residual reduction          = default
Choice for Inexact Newton Forcing Term              = 0
Calculate the Jacobian Numerically                  = no
Solution Relative Error Tolerance                   = 1.0e-3
Solution Absolute Error Tolerance                   = 1.0e-8

---------------- linear solver subsection ------------------

Solution Algorithm                                  = gmres
Convergence Norm                                    = 0
Preconditioner                                      = no_overlap_ilu
Polynomial                                          = LS,1
Scaling                                             = row_sum
Orthogonalization                                   = classical
Size of Krylov subspace                             = 92
Maximum Linear Solve Iterations                     = 500
Linear Solver Normalized Residual Tolerance         = 1.0e-6
-----------------------------------------------------------------
                    Chemistry Specifications
-----------------------------------------------------------------
Energy  equation source terms            = off
Species equation source terms            = off
Pressure (atmospheres)                   = 0 09210526
Thermal Diffusion                        = on
Multicomponent Transport                 = stefan_maxwell
Chemkin file                             = chem bin
Surface chemkin file                     = surf bin
Transport chemkin file                   = tran.bin
-----------------------------------------------------------------
              Enclosure Radiation Specifications
-----------------------------------------------------------------
Enclosure Radiation source terms         = off

-----------------------------------------------------------------
     Material ID Specifications
-----------------------------------------------------------------
Number of Materials                   = 1
NEWTONIAN                             = 0    "Air"
          ELEM_BLOCK_IDS              = 1
      VISCOSITY        = 1 0
      DENSITY          = 1.0

      U_INIT           = 50.0

END Material ID Specifications
```

```
--------------------------------------------------------------
     Boundary Condition Specifications
--------------------------------------------------------------
Number of Generalized Surfaces     = 0
Number of BC                       = 6

#Inlet boundary condition - uniform velocity
BC = U_BC DIRICHLET SS 4 INDEPENDENT 50  0
BC = V_BC DIRICHLET SS 4 INDEPENDENT 0 0 0

# Upper solid plate - No slip
BC = U_BC DIRICHLET SS 1 INDEPENDENT 0.0 0
BC = V_BC DIRICHLET SS 1 INDEPENDENT 0 0 0

# Outflow boundary condition (no normal stress on x
# component of the momentum equation)
BC = V_BC DIRICHLET SS 2 INDEPENDENT 0.0 0

# Lower boundary is on the channel centerline
# Set zero V velocity, no shear stress for U velo
BC = V_BC DIRICHLET SS 3 INDEPENDENT 0.0 0

--------------------------------------------------------------
        Initial Guess/Condition Specifications
--------------------------------------------------------------
Set Initial Condition/Guess                 = EXOII_FILE
Apply function                              = no
Time Index to Restart From                  = 1
--------------------------------------------------------------
                   Output Specifications
--------------------------------------------------------------
User Defined Output                                = yes
Parallel Output                                    = no
Scalar Output                                      = yes
Time Index to Output To                            = 1
Nodal variable output times
        every 2 steps

Number of nodal output variables                   = 2
Nodal variable names
        Velocity
        Pressure

Number of global output variables                  = 0
Global variable names

Test Exact Solution Flag                           = 0
Name of Exact Solution Function                    = f_xx_yy
--------------------------------------------------------------
                  Parallel I/O section
--------------------------------------------------------------
--------------------------------------------------------------
       Data Specification for User's Functions
--------------------------------------------------------------
Number of functions to pass data to    =  8

#Call to output data along the wall (note. tau_n is printed out
as tau_x tau_y)
Function Name    =  f_ss_centroid   2
FN_DATA = INT  1
FN_DATA = STRING  x  Area P n_grad_U tau_n

#Call for time history output at channel inlet
#The data output are- time step, time, x, y, U, V, P
Function Name = time_history_line 2
FN_DATA = INT 10
FN_DATA = TABLE 2 2
   0 0  0 0
   0 0  0 5

#Call for time history output at various locations along the
channel
Function Name = time_history_line 2
FN_DATA = INT 60
FN_DATA = TABLE 2 2
   0 025  0.0
   0 025  0.5

Function Name = time_history_line 2
FN_DATA = INT 60
FN_DATA = TABLE 2 2
   0.1  0.0
   0 1  0.5

<< 6 more time_history_line data statements follow for
increasing values of x >>
```

*Figure C.6. Input file for the* Hydrodynamically Developing Flow in an Infinite Parallel Plate Channel *example problem.*

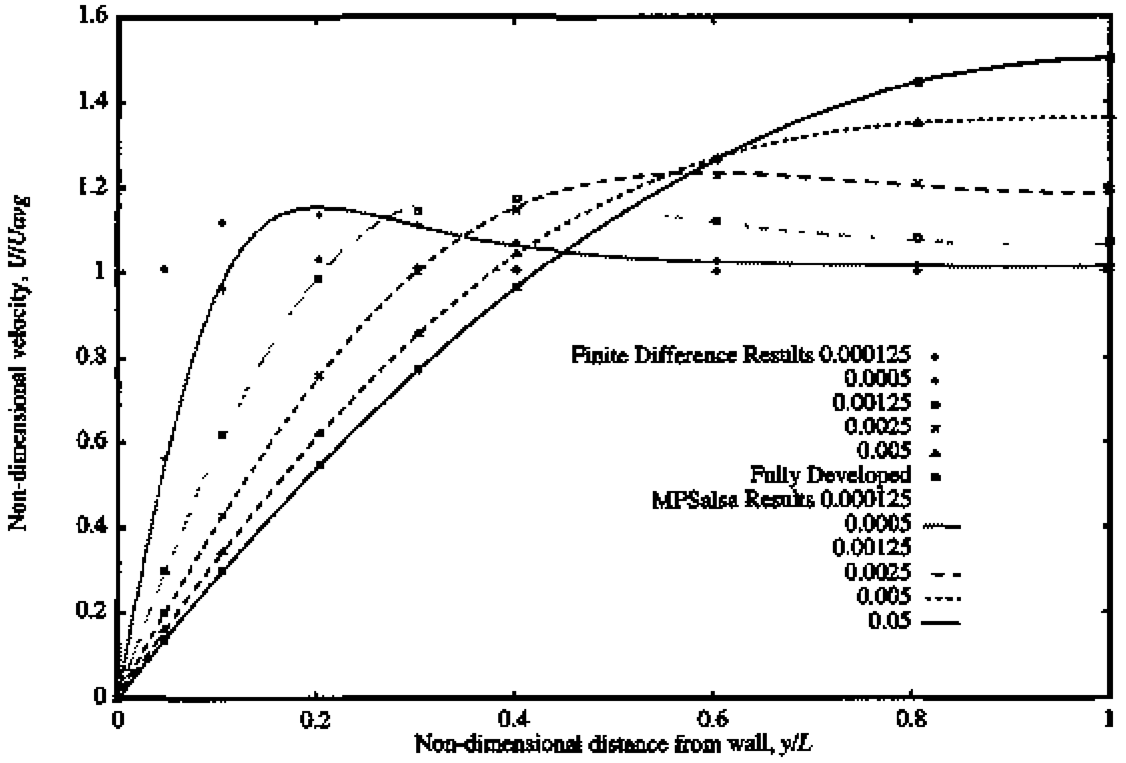


*Figure C.7. Developing velocity profiles for flow entering parallel plates for a variety of non-dimensional lengths down the channel, as the flow transitions from plug flow to a parabolic profile.*

MPSalsa input file is given in Figure C.8. The hydrodynamic boundary conditions are the same as in Appendix C.3 except for the inlet velocity boundary condition. For this condition, the function `user_bc_exact` is called. The user must program an expression for a parabolic velocity profile and place it in "rf_user_bc_exact_fn.c." For this example, the profile for the $x$-component of velocity was $6y - 6y^2$ at the inlet. For the energy equation, the Neumann boundary condition is used to set the heat flux on the solid plate; a Dirichlet boundary condition is used to set the inlet temperature level.

Reducing the temperature field data to calculate the local Nusselt numbers, the data are shown on Figure C.9 where $Nu = hD_h/k$ and $\hat{x} = x/D_h RePr$ for heat transfer coefficient $h$, thermal conductivity $k$, and half-distance between the plates $D_h$. Comparison with the three part correlation of Shah and Bhatti [46] generally were within 2% over the entire range, except where their correlation is discontinuous.

## C.5. Vortex Shedding from a Circular Cylinder

Slow flow over a cylinder yields steady solutions; however, as the Reynolds number is increased above 60, the character of laminar flow across a cylinder changes. A steady flow can no longer be maintained; rather, the flow takes on a time varying behavior with a periodic shedding

```
--------------------------------------------------------------
               General Problem Specifications
--------------------------------------------------------------
Problem type                        = fluid_flow_energy
Input FEM file                      = rect exoII
LB file                             = rect-32-bKL exoII
Output FEM file                     = rectNT-out exoII
Number of processors                = 32
Cartesian or Cylindrical when 2D    = Cartesian
Interpolation Order                 = linear
Stabilization                       = supg
Debug                               = 2
--------------------------------------------------------------
                  Solution Specifications
--------------------------------------------------------------
Solution Type                          = steady
Order of integration/continuation      = 1
Step Control                           = on
Relative Time Integration Error        = 1 0e-3
Initial Parameter Value                = 10 0
Initial Step Size                      = 30 0
Maximum Number of Steps                = 8
Maximum Time or Parameter Value        = 1 0e+2
--------------------------------------------------------------
                   Solver Specifications
--------------------------------------------------------------
Override Default Linearity Choice                 = default

---------------- nonlinear solver subsection  --------------

Number of Newton Iterations                       = 80
Use Modified Newton Iteration                     = no
Enable backtracking for residual reduction        = default
Choice for Inexact Newton Forcing Term            = 0
Calculate the Jacobian Numerically                = no
Solution Relative Error Tolerance                 = 1 0e-3
Solution Absolute Error Tolerance                 = 1 0e-8

---------------- linear solver subsection  -----------------

Solution Algorithm                                = gmres
Convergence Norm                                  = 0
Preconditioner                                    = no_overlap_ilu
Polynomial                                        = LS,1
Scaling                                           = row_sum
Orthogonalization                                 = classical
Size of Krylov subspace                           = 52
Maximum Linear Solve Iterations                   = 500
Linear Solver Normalized Residual Tolerance       = 1 0e-6
--------------------------------------------------------------
                  Chemistry Specifications
--------------------------------------------------------------
Energy  equation source terms         = off
Species equation source terms         = off
Pressure (atmospheres)                = 0 03210526
Thermal Diffusion                     = on
Multicomponent Transport              = stefan_maxwell
Chemkin file                          = chem bin
Surface chemkin file                  = surf bin
Transport chemkin file                = tran bin
--------------------------------------------------------------
             Enclosure Radiation Specifications
--------------------------------------------------------------
Enclosure Radiation source terms      = off

--------------------------------------------------------------
      Material ID Specifications
--------------------------------------------------------------
Number of Materials                 = 1
NEWTONIAN                           = 0    "Air"
        ELEM_BLOCK_IDS              = 1
     VISCOSITY        = 0 02
     DENSITY          = 1 0
     THERMAL_CONDUCT  = 0 02
     CP               = 10 0
     T_INIT           = 0 0
END Material ID Specifications
```

```
--------------------------------------------------------------
      Boundary Condition Specifications
--------------------------------------------------------------
Number of Generalized Surfaces    = 0
Number of BC                      = 8

# Lower solid plate - No slip  heat flux set at -10 0
BC = U_BC DIRICHLET SS 1 INDEPENDENT 0 0 0
BC = V_BC DIRICHLET SS 1 INDEPENDENT 0 0 0
BC = T_BC NEUMANN   SS 1 INDEPENDENT -10 0 0

#Inlet boundary condition - uniform velocity
#Note  average velocity set in rf_user_bc_exact_fn c is 1 0
with parabolic velocity profile
BC = U_BC DIRICHLET SS 4 INDEPENDENT user_bc_exact 0
BC = V_BC DIRICHLET SS 4 INDEPENDENT 0 0 0

# Inlet boundary condition - temperature
BC = T_BC DIRICHLET SS 4 INDEPENDENT 0 0 0

# Outflow boundary condition (no normal stress on x
# component of the momentum equation)
BC = V_BC DIRICHLET SS 2 INDEPENDENT 0 0 0

# Upper boundary is on the channel centerline
# Set zero V velocity, no shear stress for U velo,
# and no heat flux for temp
BC = V_BC DIRICHLET SS 3 INDEPENDENT 0 0 0

--------------------------------------------------------------
        Initial Guess/Condition Specifications
--------------------------------------------------------------
Set Initial Condition/Guess                 = constant 0 0
Apply function                              = no
Time Index to Restart From                  = 1
--------------------------------------------------------------
                  Output Specifications
--------------------------------------------------------------
User Defined Output                                = yes
Parallel Output                                    = no
Scalar Output                                      = yes
Time Index to Output To                            = 1
Nodal variable output times
        every 2 steps

Number of nodal output variables                   = 3
Nodal variable names
        Velocity
        Pressure
        Temperature

Number of global output variables                  = 0
Global variable names

Test Exact Solution Flag                           = 0
Name of Exact Solution Function                    = f_xx_yy
--------------------------------------------------------------
                 Parallel I/O section
--------------------------------------------------------------
Machine                                     = paragon
Staged writes                               = yes
------------------
paragon subsection
------------------
Number of RAID controllers                  = 26
Root location                               = /pfs/io_
Subdirectory                                = tmp/aus/ti43
Offset numbering from zero                  = 23
--------------------------------------------------------------
      Data Specification for User s Functions
--------------------------------------------------------------
Number of functions to pass data to    =  2

#Call to output data along the wall
Function Name   =  f_ss_centroid   2
FN_DATA = INT  1
FN_DATA = STRING  x  Area T n_grad_T n_grad_U


#Call for time history output at channel inlet
#The data output are  time step  time  x  y  U, V  P  T
Function Name = time_history_line 2
FN_DATA = INT 10
FN_DATA = TABLE 2 2
   0 0  0 0
   0 0  0 5
```

*Figure C.8. Input file for the* Thermally Developing Flow in an Infinite Parallel Plate Channel *example problem.*

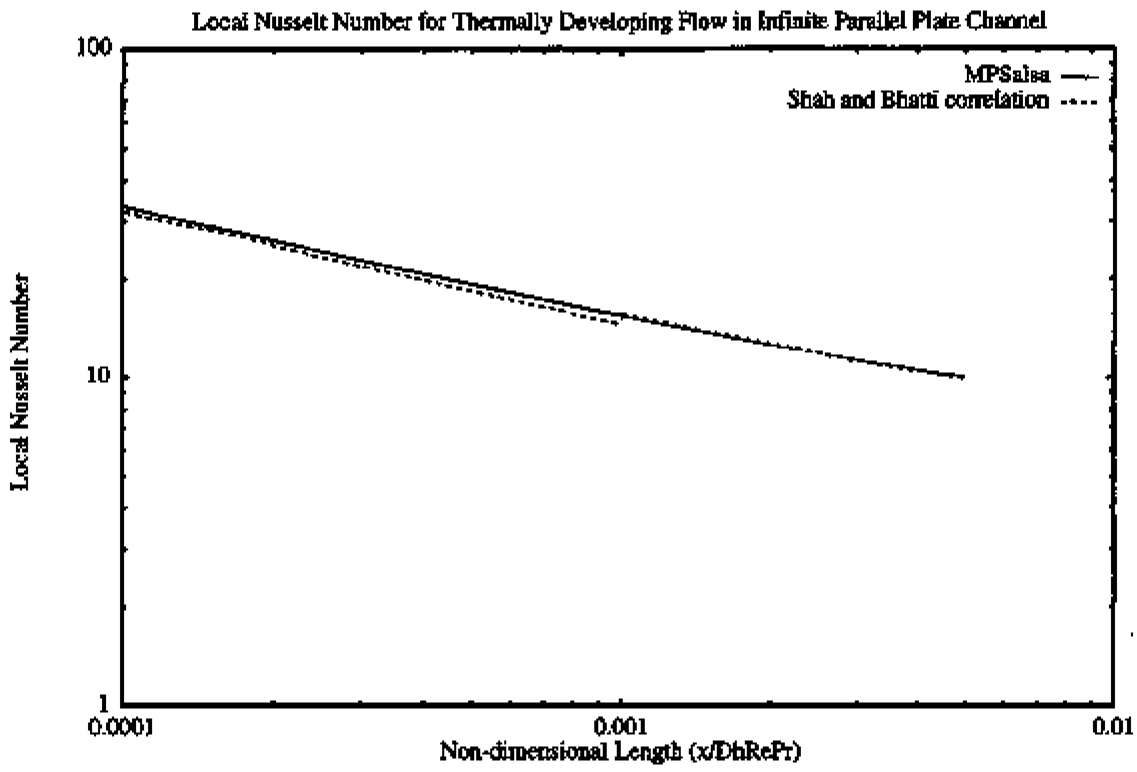


*Figure C.9. Comparison of the MPSalsa calculation and an established correlation for the Nusselt number for thermally-developing flow in a parallel plate channel*

of vortices [19]. This transient behavior is illustrated in this example. The 2D mesh consists of 4300 elements -- 80 elements around the circumference and 50 expanding away from the cylinder. The domain is shown in Figure C.10, with a channel width of 30 diameters. The circumference of the cylinder is designated Side Set 1; the two channel walls are Side Set 2; the inlet is Side Set 3; and the outflow boundary is Side Set 4.

A uniform velocity profile is provided at the inlet to the channel. The channel walls' boundary conditions are no shear and impervious. The cylinder's boundary conditions are no slip and impervious. No shear is set at the outflow boundary. Experiments with Reynolds numbers $Re = 60$, 100, 200, and 600 were done. The input file for $Re = 600$ is given in Figure C.11.

To indicate the transient nature of the flow, the time varying variables were recorded at a location a distance 4.0 downstream from the cylinder and 0.5 from the line of symmetry using the `time_history_point` function. The calculation for $Re = 60$ was started from an initial guess of zero. For higher Reynolds numbers, the calculations were started using the restart option; the solution for the next lower Reynolds number was used as the starting point. At all times, the automatic time step control was set to `on`. Care must be used in setting the initial time step size, `Relative Time Integration Error`, and `Solution Relative Error Tolerance`; values that are too large can result in the transient being missed.

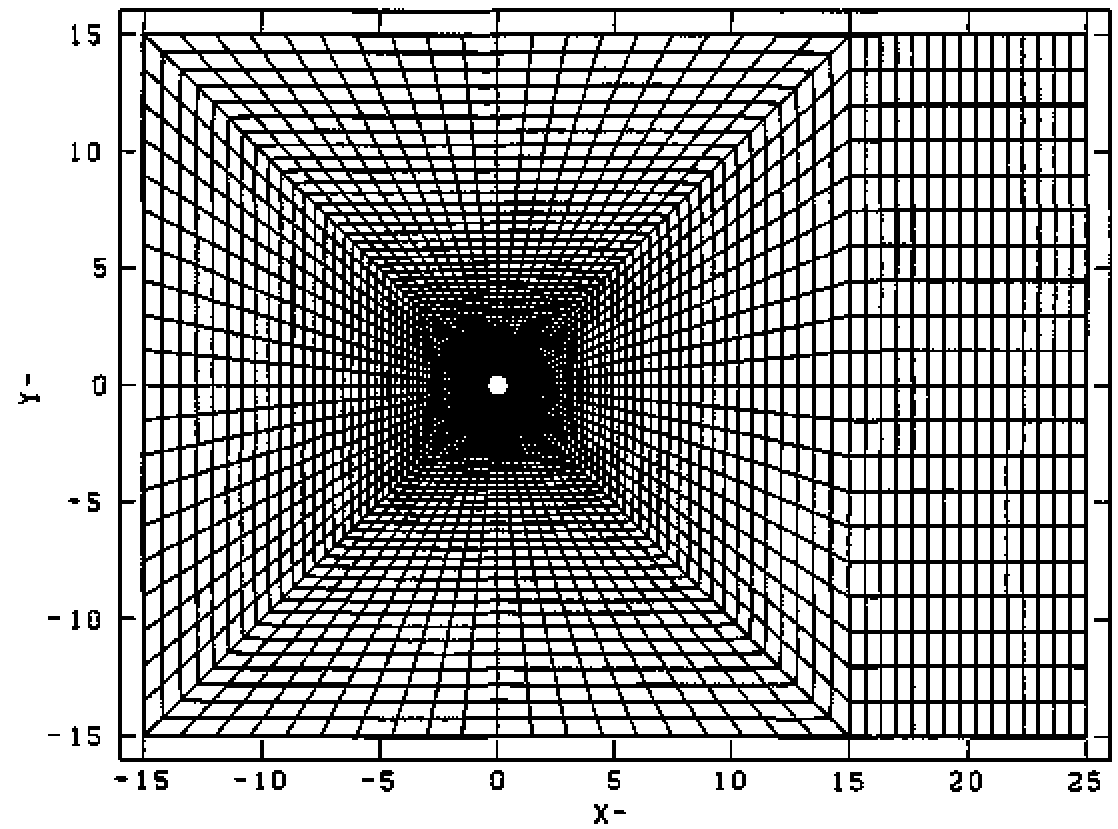


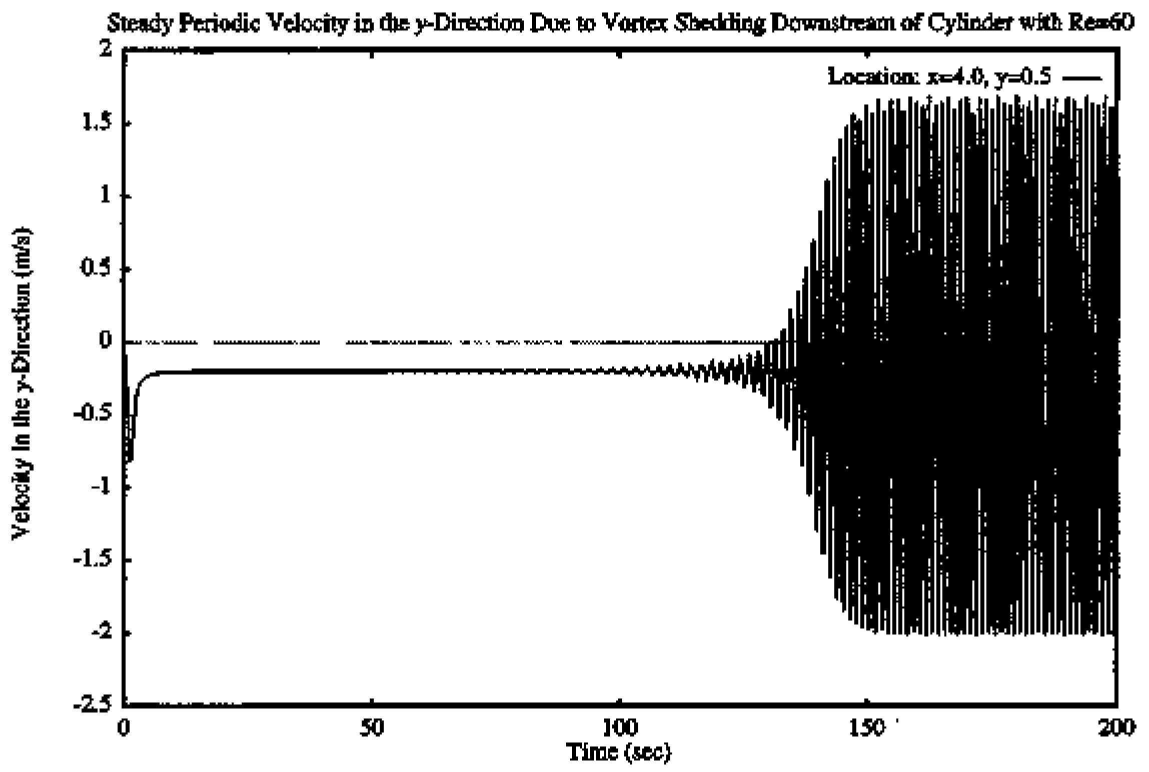
*Figure C.10. The finite element mesh of 4300 elements for the* Vortex Shedding from a Circular Cylinder *example problem.*

Shown in Figure C.12(b) is the $y$-component of velocity as a function of time for the flow with $Re = 600$. (Density was set to 1.0 and viscosity was set to 0.1 in this example, so for $Re = 600$, the average $x$-component of velocity was 60.) Figure C.12(a) shows a similar trace for $Re = 60$. The von Karman vortex street behind the cylinder with $Re = 600$ is shown in Figure C.13. In Figure C.12(a) and (b), the transient behavior before the steady periodic nature of the flow is fully established depends upon the grid geometry, convergence criteria, and initial condition. For the fully-developed, steady, periodic flow, the frequency of vortex shedding can be characterized by the non-dimensional Strouhal number, $St = fD/V$, where $f$ is the frequency of shedding, $D$ is the cylinder diameter, and $V$ is the fluid approach velocity. $St$ is a function of Reynolds number. For the flows calculated with MPSalsa, the results are shown in Table C.2. Comparison is made against experimental data presented in Schlichting [39].

```
-----------------------------------------------------------------
              General Problem Specifications
-----------------------------------------------------------------
Problem type                          = fluid_flow
Input FEM file                        = cyl exoII
LB file                               = cyl-8-LBL exoII
Output FEM file                       = cyl-Re600-out exoII
Number of processors                  = 8
Cartesian or Cylindrical when 2D      = Cartesian
Interpolation Order                   = linear
Stabilization                         = supg
Debug                                 = 2
-----------------------------------------------------------------
                 Solution Specifications
-----------------------------------------------------------------
Solution Type                           = transient
Order of integration/continuation       = 2
Step Control                            = on
Relative Time Integration Error         = 1 0e-4
Initial Parameter Value                 = 10 0
Initial Step Size                       = 0 05
Maximum Number of Steps                 = 1000
Maximum Time or Parameter Value         = 500 0
-----------------------------------------------------------------
                  Solver Specifications
-----------------------------------------------------------------
Override Default Linearity Choice                 = default

---------------- nonlinear solver subsection  --------------

Number of Newton Iterations                       = 15
Use Modified Newton Iteration                     = no
Enable backtracking for residual reduction        = no
Choice for Inexact Newton Forcing Term            = 4
Calculate the Jacobian Numerically                = no
Solution Relative Error Tolerance                 = 1 0e-4
Solution Absolute Error Tolerance                 = 1 0e-8

---------------- linear solver subsection  -----------------

Solution Algorithm                                = gmres
Convergence Norm                                  = 0
Preconditioner                                    = no_overlap_ilu
Polynomial                                        = LS,7
Scaling                                           = row_sum
Orthogonalization                                 = classical
Size of Krylov subspace                           = 80
Maximum Linear Solve Iterations                   = 200
Linear Solver Normalized Residual Tolerance       = 5 0e-4
-----------------------------------------------------------------
                 Chemistry Specifications
-----------------------------------------------------------------
Energy  equation source terms           = off
Species equation source terms           = off
Pressure (atmospheres)                  = 0 09210526
Thermal Diffusion                       = on
Multicomponent Transport                = stefan_maxwell
Chemkin file                            = chem bin
Surface chemkin file                    = surf bin
Transport chemkin file                  = tran bin
-----------------------------------------------------------------
           Enclosure Radiation Specifications
-----------------------------------------------------------------
Enclosure Radiation source terms        = off

-----------------------------------------------------------------
     Material ID Specifications
-----------------------------------------------------------------
Number of Materials                 = 1
NEWTONIAN                           = 0    "Air"
         ELEM_BLOCK_IDS             = 1
      VISCOSITY          = 0 1
      DENSITY            = 1 0

      U_INIT             = 60 0
      V_INIT             = 0 0

END Material ID Specifications
```

```
-----------------------------------------------------------------
     Boundary Condition Specifications
-----------------------------------------------------------------
Number of Generalized Surfaces    = 0
Number of BC                      = 5

# Inlet boundary condition - uniform velocity
BC = U_BC DIRICHLET NS 3 INDEPENDENT 60 0 0
BC = V_BC DIRICHLET NS 3 INDEPENDENT 0 0 0

# Cylinder - No slip
BC = U_BC DIRICHLET NS 1 INDEPENDENT 0 0 0
BC = V_BC DIRICHLET NS 1 INDEPENDENT 0 0 0

# Outflow boundary condition (no normal stress on x
# component of the momentum equation)

# Solid plates - No shear
BC = V_BC DIRICHLET NS 2 INDEPENDENT 0 0 0


-----------------------------------------------------------------
        Initial Guess/Condition Specifications
-----------------------------------------------------------------
Set Initial Condition/Guess                  = EXOII_FILE
Apply function                               = no
Time Index to Restart From                   = 805
-----------------------------------------------------------------
                 Output Specifications
-----------------------------------------------------------------
User Defined Output                                   = yes
Parallel Output                                       = no
Scalar Output                                         = yes
Time Index to Output To                               = 1
Nodal variable output times
        every 1 steps

Number of nodal output variables                      = 2
Nodal variable names
        Velocity
        Pressure

Number of global output variables                     = 0
Global variable names

Test Exact Solution Flag                              = 0
Name of Exact Solution Function                       = f_xx_yy
-----------------------------------------------------------------
               Parallel I/O section
-----------------------------------------------------------------
Machine                                      = paragon
Staged writes                                = yes
----------------
ncube subsection
----------------
Number of controllers                        = 8
Disks per controller                         = 1
Root location                                = //df
Subdirectory                                 = jns/tests
Offset numbering from zero                   = 0
------------------
paragon subsection
------------------]
Number of RAID controllers                   = 26
Root location                                = /pfs/io_
Subdirectory                                 = tmp/ags/ti43
Offset numbering from zero                   = 23
-----------------------------------------------------------------
      Data Specification for User's Functions
-----------------------------------------------------------------
Number of functions to pass data to    =  1

Function Name  =  time_history_points 1
#
FN_DATA =  TABLE 1 2
  4 0  0 5
```

*Figure C.11. Input file for the* Vortex Shedding from a Circular Cylinder *example problem,* Re=600.



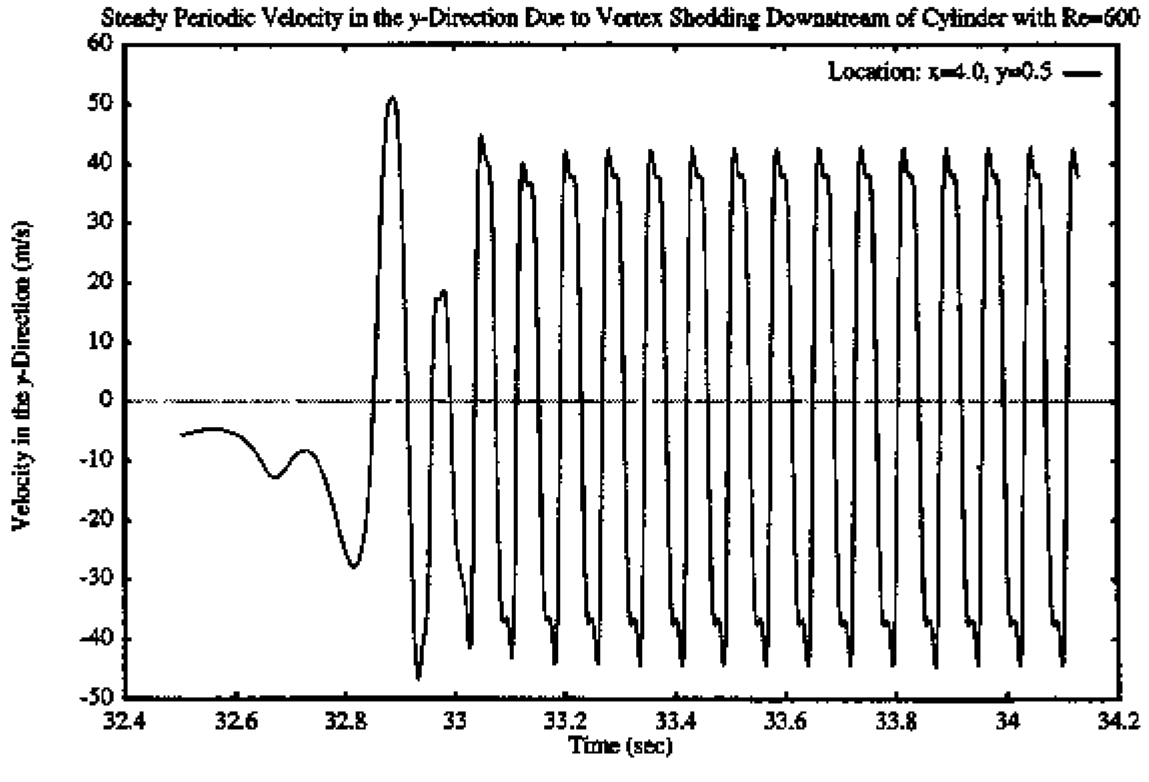


*Figure C.12. Time history plots for vortex shedding behind a cylinder: (a)* Re=60, *(b)* Re=600.

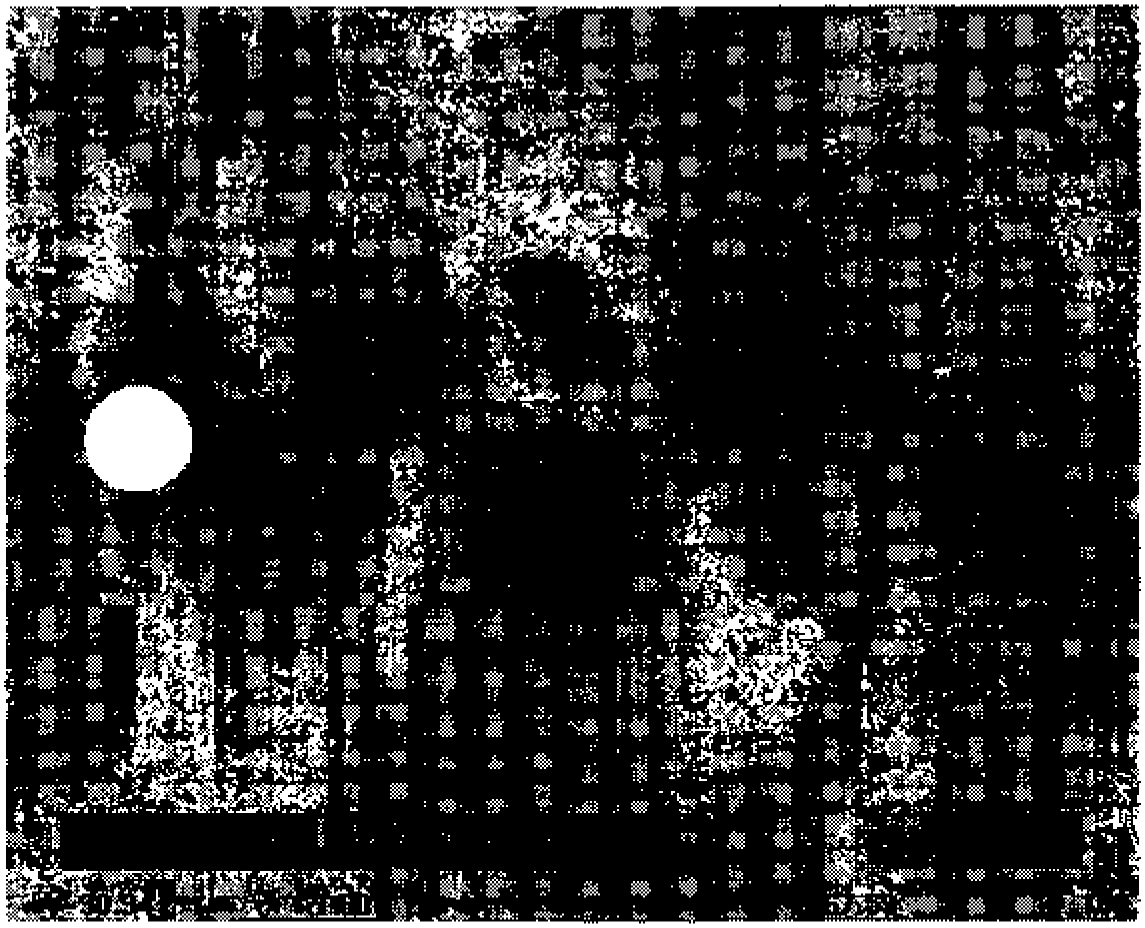


*Figure C.13. Contour plot showing the shedding vortices behind a cylinder at* Re=600.

| *Re* | *St* (MPSalsa) | *St* (Schlichting) |
|---|---|---|
| 60 | 0.132 | 0.133 |
| 100 | 0.163 | 0.166 |
| 200 | 0.189 | 0.190 |
| 600 | 0.218 | 0.210 |

*Table C.2. Comparison of Strouhal numbers as a function of Reynolds number for MPSalsa and the experimental data of Schlichting [39].*

# Appendix D. CVD Reactor Examples

## D.1. SPIN Comparison

This example problem was used to benchmark many of the capabilities of MPSalsa by comparing results with another code, SPIN [6]. SPIN solves for reacting flows in the idealized geometry of uniform flow impinging on a rotating disk of infinite radius, by using the von Karman similarity solution that reduces the 3D problem to 1D. We solve a full 3D problem using MPSalsa of flow impinging on a rotating disk with large radius, and compare the solutions near the center of the disk with SPIN. The excellent agreement between the two solutions verifies our implementation of the fluid mechanics, heat and mass transfer, gas-phase reactions, surface reactions, and the Danckwerts' boundary conditions.

Our computational domain for the MPSalsa calculation is cylindrical, with an inlet at 10cm above a reactive rotating disk with a radius of 7cm. The surface of the 12,660-element mesh used in this calculation, generated using CUBIT [24], is shown in Figure D.1.

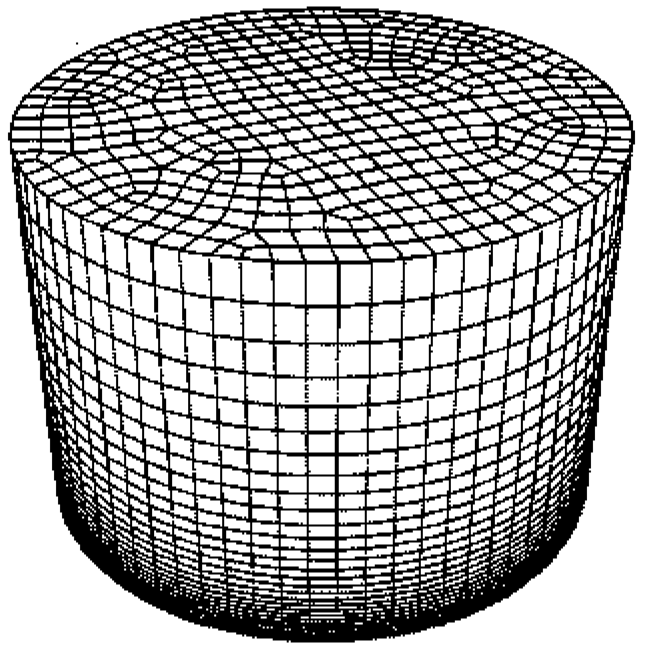


*Figure D.1. Surface of 12,660-element mesh for* SPIN Comparison *example problem.*

The reaction mechanism used in this calculation is for the deposition of Silicon, and has 8 gas-phase species, 10 gas-phase reactions, 2 surface species, 8 surface reactions, and 1 bulk component (solid silicon). A schematic diagram of the system is shown in Figure D.2.

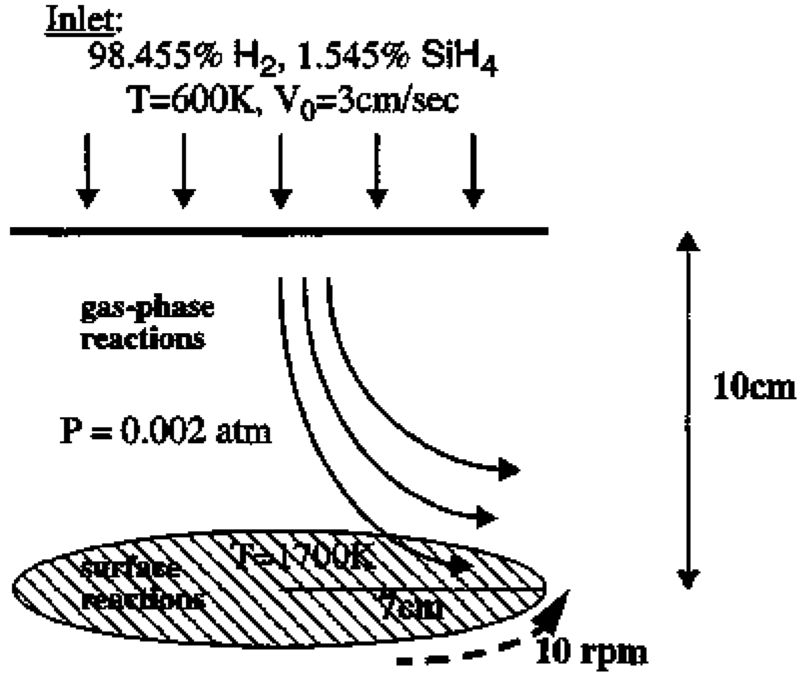


*Figure D.2. Schematic diagram of* SPIN Comparison *example problem. Plug flow enters the low pressure reactor 10 cm above a heated disk with radius 7cm that is rotating at 10 rpm. Gas-phase reactions and surface reactions proceed as a function of concentrations and temperature.*

Since the system is operating at a low pressure of 0.002 atmospheres, the diffusive flux of species at the inlet boundary of the computational domain is non-negligible. In experiments, it is the total flux of each species into the domain that is known, but setting Dirichlet conditions for the species mole fractions and inlet velocity sets only the convective flux while ignoring the diffusive contribution. Danckwerts' boundary condition allows for the specification of the total flux at the inlet boundary of the computational domain, and functions are included in MPSalsa to implement this condition (see Section A.1.2).

The input file for this example problem is shown in Figure D.3. The problem is run on 256 processors, and can reach the steady-state directly using the `tfqmr` linear solver with `no_overlap_bilu` preconditioning. Danckwerts' boundary condition on the velocity and species mole fractions is specified at the inlet (side set 1), and surface reactions and spinning conditions are specified on the disk surface (side set 2). The output function `time_history_line` is used to print information along a vertical line at radius 1cm, as specified at the bottom of the input file.

The 3D steady state was reached in 10 minutes on 256 Processors of the Intel Paragon, and required 7 Newton iterations and 1149 total iterations of the linear solver. Solving the analogous infinite disk problem with SPIN required only 20 seconds on a workstation. The

```
-----------------------------------------------------------------
               General Problem Specifications
-----------------------------------------------------------------
Problem type                          = whole_enchilada
Input FEM file                        = Meshes/si_13k exoII
LB file                               = Meshes/si_13k-256-LBL exoII
Output FEM file                       = run_out exoII
Number of processors                  = 256
Cartesian or Cylindrical when 2D      = Cartesian
Interpolation Order                   = linear
Stabilization                         = default
Debug                                 = 2
-----------------------------------------------------------------
                 Solution Specifications
-----------------------------------------------------------------
Solution Type                           = steady
Order of integration/continuation       = 1
Step Control                            = off
Relative Time Integration Error         = 4 0e-3
Initial Parameter Value                 = 10 0
Initial Step Size                       = 30 0
Maximum Number of Steps                 = 5
Maximum Time or Parameter Value         = 1 0e+2
-----------------------------------------------------------------
                 Solver Specifications
-----------------------------------------------------------------
Override Default Linearity Choice                  = default

---------------- nonlinear solver subsection  --------------

Number of Newton Iterations                        = 15
Use Modified Newton Iteration                      = no
Enable backtracking for residual reduction         = no
Choice for Inexact Newton Forcing Term             = 4
Calculate the Jacobian Numerically                 = no
Solution Relative Error Tolerance                  = 1 0e-3
Solution Absolute Error Tolerance                  = 1 0e-6

---------------- linear solver subsection  ----------------

Solution Algorithm                                 = tfqmr
Convergence Norm                                   = 1
Preconditioner                                     = no_overlap_bilu
Polynomial                                         = LS 1
Scaling                                            = row_sum
Orthogonalization                                  = classical
Size of Krylov subspace                            = 200
Maximum Linear Solve Iterations                    = 600
Linear Solver Normalized Residual Tolerance        = 1 0e-4
-----------------------------------------------------------------
                 Chemistry Specifications
-----------------------------------------------------------------
Energy  equation source terms         = on
Species equation source terms         = on
Pressure (atmospheres)                = 0 002
Thermal Diffusion                     = on
Multicomponent Transport              = stefan_maxwell
Chemkin file                          = chem bin
Surface chemkin file                  = surf bin
Transport chemkin file                = tran bin
-----------------------------------------------------------------
            Enclosure Radiation Specifications
-----------------------------------------------------------------
Enclosure Radiation source terms      = off
-----------------------------------------------------------------
         Material ID Specifications
-----------------------------------------------------------------
Number of Materials                 = 1
CHEMKIN                             = 0  "silicon"
         ELEM_BLOCK_IDS             = 7 9
     T_INIT                         = 600
# Change from U1  U2  U3
     U_INIT                         = 0 0
     V_INIT                         = 0 0
     W_INIT                         = -2 0

         XMF_0  H2       0 9995
         XMF_0  SiH4     0 0005
END Material ID Specifications
```

```
-----------------------------------------------------------------
        Boundary Condition Specifications
-----------------------------------------------------------------
Number of Generalized Surfaces = 0

Number of BC                          = 10
BC    = T_BC DIRICHLET SS 1 INDEPENDENT  600 0 0
BC    = T_BC DIRICHLET SS 2 INDEPENDENT 1700 0 0
#
BC    = U_BC DIRICHLET SS 1 INDEPENDENT 0   0
BC    = U_BC DIRICHLET SS 2 INDEPENDENT f_xy_spin_disk 1
        BC_DATA = 10 0 0 0 0 0
#
BC    = V_BC DIRICHLET SS 1 INDEPENDENT 0 0 0
BC    = V_BC DIRICHLET SS 2 INDEPENDENT f_xy_spin_disk 1
        BC_DATA = 10 0 0 0 0 0
#
BC    = W_BC DIRICHLET SS 1 DEPENDENT f_Danckwerts_X0 1
        BC_DATA =  -3 0 0 01545 0 0 0 0 0 0 0 0 0 0 0 0 98455
BC    = W_BC DIRICHLET SS 2 DEPENDENT surface_chemkin_bc 0
#
BC    = Y_BC MIXED  SS 1  INDEPENDENT f_Danckwerts
f_Danckwerts_X0 0 0 1
        SPECIES_LIST =  ALL
        BC_DATA =  -3 0 0 01545 0 0 0 0 0 0 0 0 0 0 0 0 98455
BC    = Y_BC NEUMANN   SS 2 DEPENDENT surface_chemkin_bc 0
        SPECIES_LIST = ALL
#
-----------------------------------------------------------------
        Initial Guess/Condition Specifications
-----------------------------------------------------------------
Set Initial Condition/Guess                   = constant 0 0
Apply function                                = no
Time Index to Restart From                    = 1
-----------------------------------------------------------------
                  Output Specifications
-----------------------------------------------------------------
User Defined Output                                  = yes
Parallel Output                                      = no
Scalar Output                                        = yes
Time Index to Output To                              = 1
Nodal variable output times
        every 2 steps

Number of nodal output variables                     = 4
Nodal variable names
        Temperature
        Velocity
        Pressure
        Mass_fraction

Number of global output variables                    = 0
Global variable names

Test Exact Solution Flag                             = 0
Name of Exact Solution Function                      = f_xx_yy
-----------------------------------------------------------------
                Parallel I/O section
-----------------------------------------------------------------
Machine                                       = paragon
Staged writes                                 = yes

-------------------
paragon subsection
-------------------
Number of RAID controllers                    = 26
Root location                                 = /pfs/io_
Subdirectory                                  = tmp/aga/ti43
Offset numbering from zero                    = 33
-----------------------------------------------------------------
       Data Specification for User's Functions
-----------------------------------------------------------------
Number of functions to pass data to    =  1

Function Name      =    time_history_line 2
#
FN_DATA = INT 100
FN_DATA = TABLE 2 3
   0 6 0 8 10 0
   0 6 0 8  0 0
```

*Figure D.3. Input file for the* SPIN Comparison *example problem.*

adaptive gridding strategy placed 171 nodes in the 1D mesh, as compared to the 30 elements in the axial direction of the 3D MPSalsa mesh.

Comparisons between MPSalsa and SPIN can be seen in Figure D.4. Excellent agreement can be seen for all quantities except the axial velocity, for which the differences reflect the fact that SPIN is solving the problem on an infinite domain while MPSalsa uses a finite domain. The axial velocity in the MPSalsa calculation is strongly effected by the boundaries of the computational domain at finite radius. The discrepancy diminishes at higher flow rates. The Stefan velocity into the disk does agree between the calculations, and is uncommonly large because of the huge difference in molecular weights between Si and $H_2$ and the low operating pressure.

## D.2. Rotating Disk Reactor

A real reactor used for the growth of Gallium Arsenide single crystals is the rotating disk reactor [2, 12]. The reactor is designed to capitalize on the perfect uniformity of deposition of the infinite disk configuration, with the plug flow of reactants impinging on a rotating disk. The reactor geometry, shown in Figure D.5, consists of a vertical cylinder sitting concentrically inside a larger cylindrical reaction vessel. Flow enters uniformly through the circular cross-section of the reactor and the inner cylinder is rotated, with the reaction occurring on the top heated surface. Flow exits through the annular region between the cylinders. Very uniform growth has been observed in this reactor over a large central section of the disk where the effects of a finite radius system are small.

The reaction mechanism used in this system for chemical vapor deposition of Gallium Arsenide (from Moffat et al. [35]) consists of 4 gas-phase species, 3 surface species, and 2 bulk species, and can be found in the Chemkin input files "gaas_block.gas" and "gaas_block.sur." There are no gas-phase reactions, and 3 surface reactions.

In this example problem, we demonstrate the restarting capability in MPSalsa by solving for three different steady states of the reactor at three different sets of operating parameters, as presented in Table D.1. The solution at the first set of conditions is used as the initial guess for finding the steady state at the second set, since it is closer to the solution than a trivial initial guess. Similarly, the third solution uses the second solution as an initial guess. Being able to restart from a previous solution is necessary for reactor analysis, where many sets of operating conditions need to be explored. Also, using a series of steady-state jumps can be an efficient way of reaching a solution at conditions that are too complicated to allow convergence from a trivial initial guess.

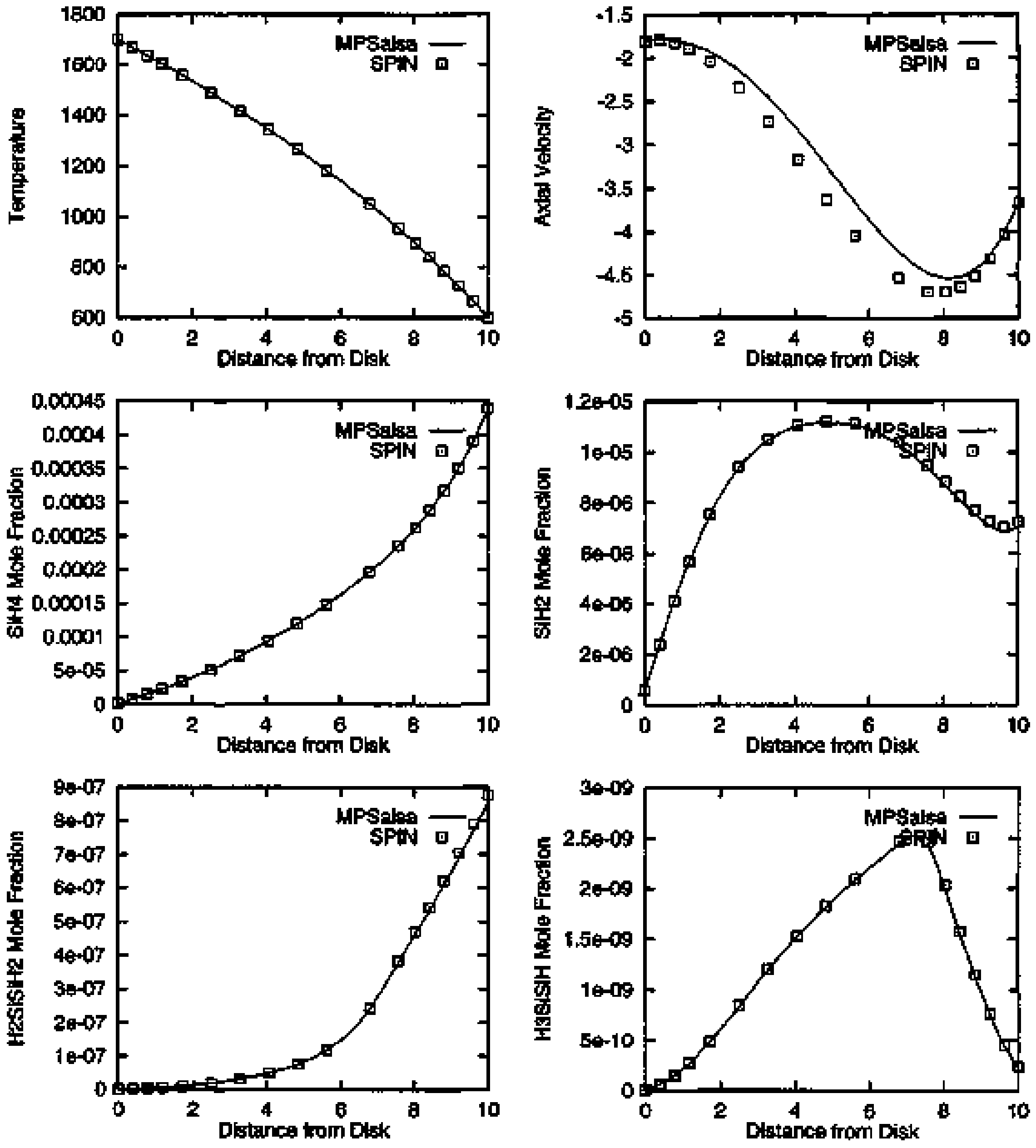


*Figure D.4. Comparisons between MPSalsa and SPIN for reacting flow impinging on an infinite rotating disk. Axial profiles of several quantities are plotted: Temperature, Axial Velocity, and Mole Fractions of $SiH_4$, $SiH_2$, $H_2SiSiH_2$ and $H_3SiSiH$.*

The input file used to solve for the steady-state at the second set of conditions in Table D.1 (using the solution at the first set of conditions in as the initial guess) can be seen in Figure D.6. In the Initial Guess/Condition Specifications section, the lines

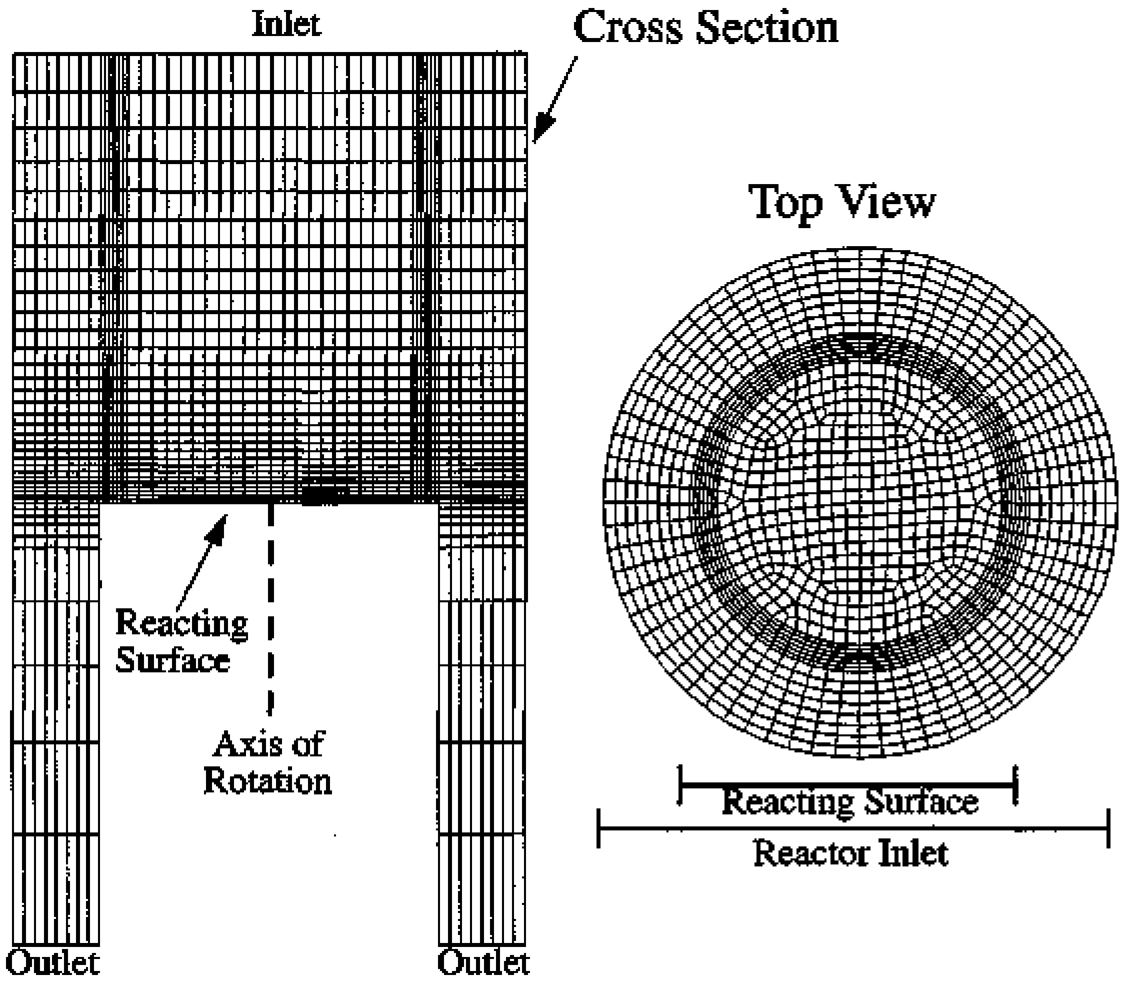


*Figure D.5. A cross section and top view of the geometry for the* Rotating Disk Reactor *example problem, showing a refined mesh. The design consists of one cylinder inside a larger one, with the reacting surface on the top of the inner cylinder, which is usually rotating. The flow enters uniformly within the entire top circle, flows over the disk, and flows out through an annular region.*

| Solution Number | Disk Spin Rate (rpm) | Inlet Flow Velocity (cm/sec) | Inlet Mole Fraction of $GaMe_3$ |
|---|---|---|---|
| 1 | 50 | 5 | 0.00013 |
| 2 | 100 | 15 | 0.00013 |
| 3 | 100 | 15 | 0.00065 |

*Table D.1. Three sets of conditions for three runs of the* Rotating Disk Reactor *example problem.*

```
Set Initial Condition/Guess              = EXOII_FILE
Time Index to Restart From               = 1
```

control the restarting. The keyword `EXOII_FILE` tells MPSalsa to get the initial guess from the output file, which in this case is named "run-out.exoII." Since this file can store many solutions for this mesh, the second line tells MPSalsa to use the first solution. The input lines `Time Index to Output To` and `Nodal variable output times` control the solution output to the ExodusII file. When a solution is being written, the time index is echoed to the standard output so the user can keep track of which solution is stored in which location of the output file.

The boundary conditions in the input file are imposed over 6 different side sets, with SS#1 being the top circular inlet, SS#2 the annular outlet region, SS#3 being the cooled outer walls, SS#4 the heated, reactive, rotating disk, and SS#5 and SS#6 being the outside of the inner, rotating cylinder. The `f_xy_spin_disk` function is used to specify velocity boundary conditions for the rotation of the inner cylinder, with the `BC_DATA` statement following it supplying the rotation rate (in rpm) and the $(x, y)$ center of rotation. The `surface_chemkin_bc` boundary condition uses the surface reaction information to specify the mass flux of each species to the surface as well as the velocity into it (see Appendix A.1.1). The `f_mole_fraction` boundary condition is used to specify the mole fractions of species at the inlet, as opposed to the mass fractions that are the primitive variables (see Appendix A.1.4). The `SPECIES_LIST` information is used to match up the input with the order that the species are in the Chemkin input file. (Since the `SPECIES_LIST` has "1" as the first entry, 0.0044 is the specified mole fraction for the first species in the Chemkin input file, which is $AsH_3$ in this case. The `SPECIES_LIST` can be listed as species names instead of integers to reduce possible confusion.) The `f_pressure` boundary condition is an outflow boundary condition that matches the normal component of the normal stress with the local pressure (see Appendix A.1.5).

With fluid mechanics and heat transfer, there are a total of 9 unknowns per node. For the coarse mesh of 7472 elements and 8499 nodes used in this example problem, this corresponds to 76,491 total unknowns. (Published results for this reactor use a much finer mesh of around 40,000 elements [2, 12].) The problem is solved on 64 processors of the Intel Paragon.

Table D.2 shows some solution statistics for the three solutions. The number of Newton iterations and the solution time for the second and third solutions were less than those of the first solution -- even though they were at more difficult parameter values -- because the initial guess from a previous solution was used.

| Solution Number | Initial Guess | # of Newton Iterations | # of GMRES Iterations | Execution Time on 64 Processors |
|---|---|---|---|---|
| 1 | Trivial | 10 | 863 | 510 sec |
| 2 | Solution 1 | 8 | 904 | 459 sec |
| 3 | Solution 2 | 6 | 637 | 336 sec |

*Table D.2. Solution statistics for the three solutions for the* Rotating Disk Reactor *example problem. The parameter values are shown in Table D.1. Restarting from the previous solution decreased the execution time.*

```
-----------------------------------------------------------------
              General Problem Specifications
-----------------------------------------------------------------
Problem type                        = whole_enchilada
Input FEM file                      = run-out.exoII
LB file                             = Meshes/cm_7k-64-bKL.exoII
Output FEM file                     = run-out.exoII
Number of processors                = 64
Cartesian or Cylindrical when 2D    = Cartesian
Interpolation Order                 = linear
Stabilization                       = default
Debug                               = 2
-----------------------------------------------------------------
                  Solution Specifications
-----------------------------------------------------------------
Solution Type                          = steady
Order of integration/continuation      = 1
Step Control                           = off
Relative Time Integration Error        = 4.0e-3
Initial Parameter Value                = 10.0
Initial Step Size                      = 30 0
Maximum Number of Steps                = 4
Maximum Time or Parameter Value        = 1 0e+2
-----------------------------------------------------------------
                  Solver Specifications
-----------------------------------------------------------------
Override Default Linearity Choice                = default

---------------- nonlinear solver subsection  --------------

Number of Newton Iterations                      = 15
Use Modified Newton Iteration                    = no
Enable backtracking for residual reduction       = no
Choice for Inexact Newton Forcing Term           = 4
Calculate the Jacobian Numerically               = no
Solution Relative Error Tolerance                = 1.0e-3
Solution Absolute Error Tolerance                = 1.0e-8

--------------- linear solver subsection. ---------------

Solution Algorithm                               = gmres
Convergence Norm                                 = 1
Preconditioner                                   = no_overlap_ilu
Polynomial                                       = LS,1
Scaling                                          = row_sum
Orthogonalization                                = classical
Size of Krylov subspace                          = 150
Maximum Linear Solve Iterations                  = 300
Linear Solver Normalized Residual Tolerance      = 1.0e-5
-----------------------------------------------------------------
                  Chemistry Specifications
-----------------------------------------------------------------
Energy  equation source terms        = on
Species equation source terms        = on
Pressure (atmospheres)               = 0.09210526
Thermal Diffusion                    = on
Multicomponent Transport             = stefan_maxwell
Chemkin file                         = chem.bin
Surface chemkin file                 = surf.bin
Transport chemkin file               = tran.bin
-----------------------------------------------------------------
             Enclosure Radiation Specifications
-----------------------------------------------------------------
Enclosure Radiation source terms     = off
-----------------------------------------------------------------
         Material ID Specifications
-----------------------------------------------------------------
Number of Materials                = 1
CHEMKIN                            = 0  "gaas_new"
         ELEM_BLOCK_IDS            = 1
     T_INIT                        = 500
# Change from U1, U2, U3
     W_INIT                        =-5 0
         XMF_0  AsH3       0 0044
         XMF_0 GaMe3       0.00013
         XMF_0    H2       0.99547
     G_VECTOR 0 0 0.0 -980 0
END Material ID Specifications
```

```
-----------------------------------------------------------------
         Boundary Condition Specifications
-----------------------------------------------------------------
Number of Generalized Surfaces = 0
Number of BC                           = 23
BC    = T_BC DIRICHLET SS 1 INDEPENDENT 303.15 0
BC    = T_BC DIRICHLET SS 3 INDEPENDENT 293 15 0
BC    = T_BC DIRICHLET SS 4 INDEPENDENT 913.15 0
BC    = T_BC DIRICHLET SS 5 INDEPENDENT 913.15 0
#
BC    = U_BC DIRICHLET SS 1 INDEPENDENT 0  0
BC    = U_BC DIRICHLET SS 3 INDEPENDENT 0  0
BC    = U_BC DIRICHLET SS 4 INDEPENDENT f_xy_spin_disk 1
        BC_DATA = 100 0 0  0.
BC    = U_BC DIRICHLET SS 5 INDEPENDENT f_xy_spin_disk 1
        BC_DATA = 100.0 0. 0.
BC    = U_BC DIRICHLET SS 6 INDEPENDENT f_xy_spin_disk 1
        BC_DATA = 100.0 0. 0.
#
BC    = V_BC DIRICHLET SS 1 INDEPENDENT 0  0
BC    = V_BC DIRICHLET SS 3 INDEPENDENT 0. 0
BC    = V_BC DIRICHLET SS 4 INDEPENDENT f_xy_spin_disk 1
        BC_DATA = 100.0 0. 0
BC    = V_BC DIRICHLET SS 5 INDEPENDENT f_xy_spin_disk 1
        BC_DATA = 100 0 0  0.
BC    = V_BC DIRICHLET SS 6 INDEPENDENT f_xy_spin_disk 1
        BC_DATA = 100.0 0. 0
#
BC    = W_BC DIRICHLET SS 1 INDEPENDENT -15.0 0
BC    = W_BC DIRICHLET SS 3 INDEPENDENT 0. 0
BC    = W_BC DIRICHLET SS 4 DEPENDENT surface_chemkin_bc 0
BC    = W_BC DIRICHLET SS 5 INDEPENDENT 0  0
BC    = W_BC DIRICHLET SS 6 INDEPENDENT 0. 0
BC    = W_BC NEUMANN   SS 2 DEPENDENT f_pressure 1
        BC_DATA = FLOAT -.95
##
BC    = Y_BC DIRICHLET SS 1 INDEPENDENT f_mole_fraction 1
        SPECIES_LIST = 1 2 3 4
        BC_DATA =  0.0044 0 00013 0.0 0 99547
BC    = Y_BC NEUMANN   SS 4 DEPENDENT surface_chemkin_bc 0
        SPECIES_LIST = ALL
BC    = Y_BC NEUMANN   SS 5 DEPENDENT surface_chemkin_bc 0
        SPECIES_LIST = ALL
-----------------------------------------------------------------
         Initial Guess/Condition Specifications
-----------------------------------------------------------------
Set Initial Condition/Guess                    = EXOII_FILE
Apply function                                 = no
Time Index to Restart From                     = 1
-----------------------------------------------------------------
                   Output Specifications
-----------------------------------------------------------------
User Defined Output                                   = yes
Parallel Output                                       = no
Scalar Output                                         = yes
Time Index to Output To                               = 2
Nodal variable output times:
         every 2 steps

Number of nodal output variables                      = 4
Nodal variable names:
         Temperature
         Velocity
         Pressure
         Mass_fraction

Number of global output variables                     = 0
Global variable names.

Test Exact Solution Flag                              = 0
Name of Exact Solution Function                       = f_xx_yy
-----------------------------------------------------------------
                  Parallel I/O section
-----------------------------------------------------------------
Machine                                          = paragon
Staged writes                                    = yes
------------------
paragon subsection
------------------
Number of RAID controllers                       = 16
Root location                                    = /pfs/io_
Subdirectory                                     = tmp/aga/ti43
Offset numbering from zero                       = 13
-----------------------------------------------------------------
       Data Specification for User's Functions
-----------------------------------------------------------------
Number of functions to pass data to    =  0
```

*Figure D.6. MPSalsa input file for the* Rotating Disk Reactor *example problem.*

The three steady-state solutions computed here are axisymmetric. The deposition rate of Gallium Arsenide on the reacting surface as a function of the radial position is shown in Figure D.7. An increase in velocity increases the deposition rate between solutions 1 and 2, and the increase in the reactant concentration increases the deposition rate between solutions 2 and 3. The large deposition rate at large radii is due to the rapid flow rate passing by the corner of the disk on its way out the annular exit region. Crystal is harvested only in the center 2.5 cm region where the deposition rate is more uniform.

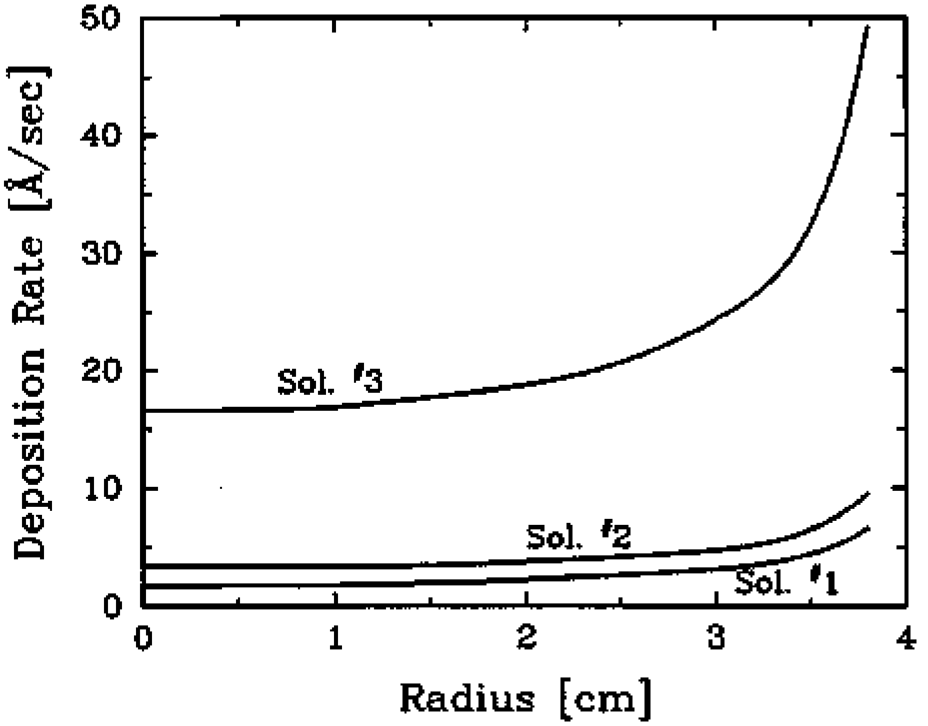


*Figure D.7. Deposition profiles of GaAs crystal in the* Rotating Disk Reactor *for three different sets of conditions (see Table D.1) as a function of the radial position on the disk.*

## D.3. Tilted Reactor

The horizontal CVD reactor with tilted susceptor and rotating substrate admits only three-dimensional solutions. This configuration is an alternative to the rotating disk reactor for growing Gallium Arsenide semiconductor crystals. We have used the same mechanism as in Appendix D.2, which includes four gas-phase species.

The reactor configuration is shown in Figure D.8. Surface reaction (deposition) occurs over the entire rectangular susceptor region, though the crystal is harvested only from the inset rotating disk. The tilted bottom of the reactor causes the flow to accelerate down the reactor length which decreases the boundary layer thickness. The increase in mass transfer to the surface due to the thinning boundary layer is in part counterbalanced by the decrease in available reactant.

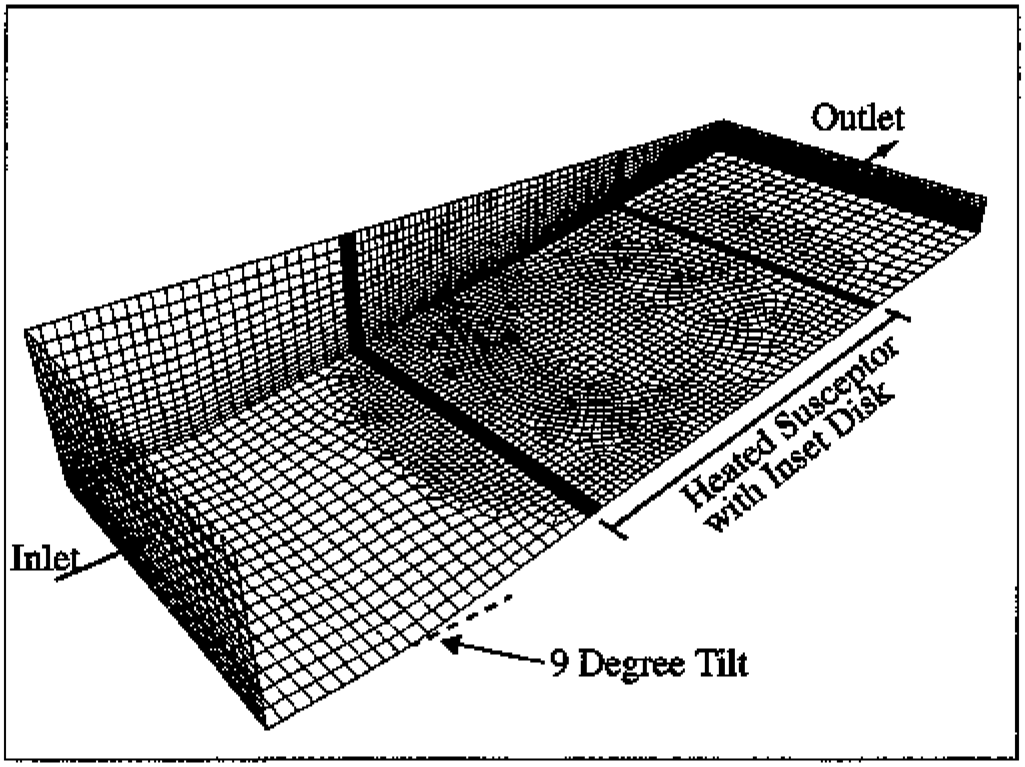


*Figure D.8. Surface mesh for the* Tilted Reactor *example problem. The hexahedral mesh consists of 43,568 elements, 48,025 nodes, and 432,225 total unknowns. A steady-state solution requires 20 minutes on 256 processors of the Intel Paragon.*

In this example problem, the continuation solution type is demonstrated. The details can be seen in the Solution Specifications section of the input file (Figure D.9), which is reproduced here.

```
Solution Type                         = continuation
Order of integration/continuation     = 1
Step Control                          = off
Initial Parameter Value               = 0.0
Initial Step Size                     = 100.0
Maximum Number of Steps               = 3
```

The above six lines tell the program, respectively, that a continuation run is to take place, that first-order continuation is to be used, that the parameter step size between solutions is to remain constant, that the initial parameter value is 0.0, that the step size is 100, and that the run will stop after three steps.

The continuation parameter itself is assigned in the file "rf_user_continuation.c," and in this case is assigned to the disk spin rate. Since the disk spin rate is supplied in the first two boundary conditions (numbered 0 and 1), and is entered as the first component (indexed 0) of the BC_DATA = FLOAT data array, the assignment of the continuation parameter to the disk spin rate requires only this line:

```
BC_Types[0].BC_Data_Float[0] = BC_Types[1].BC_Data_Float[0] = *con_par;
```

Also of note in the input file are the use of generalized surfaces and boundary condition functions. Since the disk has both velocity boundary conditions due to disk rotation in each of the tangential directions and reaction-induced flow (the Stefan velocity) in the normal direction, and since these directions do not line up with the Cartesian coordinates, generalized surfaces are needed. The function f_xy_spin_tilt9_disk (see Appendix A.1.3.2) is a special function to calculate the tangential velocities of the rotating disk as a function of the position. This function requires four arguments: the disk rotation rate (in rpm) and the coordinates of the center of the disk. The Stefan velocity is imposed using the surface_chemkin_bc as a Dirichlet condition on the normal velocity (see Appendix A.1.1).

At the end of the boundary condition section, the surface_chemkin_bc is also used to capture the effects of the surface reactions on the mass balances. In this case, we have exercised the option of providing initial guesses for the surface site and bulk fractions by use of the SURFACE_SPECIES_LIST and associated BC_DATA statements.

The GMRES linear solver was used with a Krylov subspace size of 140, which, for this problem, is the largest subspace that fits on 256 processors of the Intel Paragon at Sandia National Laboratories. The no_overlap_bilu preconditioner (incomplete block-LU decomposition without overlap between processors) was used along with row_sum scaling. A standard Newton's method was used, with backtracking turned off and a forcing term flag value of 4 to turn off the inexact Newton algorithms.

The problem was run on 256 processors of the Intel paragon. MPSalsa required 62 minutes to complete the continuation run on a mesh with 43,568 elements, 48,025 nodes, and 432,225 total unknowns. The four solutions at disk spin rates of 0, 100, 200, and 300 rpms required 12, 9, 8, and 9 Newton iterations, respectively. The first solution required more iterations because it used a trivial initial guess. The first-order continuation algorithm requires one additional matrix fill and solve after each step to calculate the tangent to the solution branch, which is used to predict an initial guess for the next step.

```
-----------------------------------------------------------
              General Problem Specifications
-----------------------------------------------------------
Problem type                        = whole_enchilada
Input FEM file                      = Meshes/ta_43k.exoII
LB file                             = Meshes/ta_43k-256-bKL.exoII
Output FEM file                     = run-out.exoII
Number of processors                = 256
Cartesian or Cylindrical when 2D    = Cartesian
Interpolation Order                 = linear
Stabilization                       = default
Debug                               = 2
-----------------------------------------------------------
                 Solution Specifications
-----------------------------------------------------------
Solution Type                          = continuation
Order of integration/continuation      = 1
Step Control                           = off
Relative Time Integration Error        = 0 0
Initial Parameter Value                = 0.0
Initial Step Size                      = 100 0
Maximum Number of Steps                = 3
Maximum Time or Parameter Value        = 1 0e+5
-----------------------------------------------------------
                  Solver Specifications
-----------------------------------------------------------
Override Default Linearity Choice                = default

---------------- nonlinear solver subsection: --------------

Number of Newton Iterations                      = 15
Use Modified Newton Iteration                    = no
Enable backtracking for residual reduction       = no
Choice for Inexact Newton Forcing Term           = 4
Calculate the Jacobian Numerically               = no
Solution Relative Error Tolerance                = 1.0e-3
Solution Absolute Error Tolerance                = 1.0e-8

---------------- linear solver subsection  -----------------

Solution Algorithm                               = gmres
Convergence Norm                                 = 1
Preconditioner                                   = no_overlap_bilu
Polynomial                                       = LS,1
Scaling                                          = row_sum
Orthogonalization                                = classical
Size of Krylov subspace                          = 140
Maximum Linear Solve Iterations                  = 280
Linear Solver Normalized Residual Tolerance      = 1.0e-3
-----------------------------------------------------------
                 Chemistry Specifications
-----------------------------------------------------------
Energy  equation source terms       = on
Species equation source terms       = on
Pressure (atmospheres)              = 0.09210526
Thermal Diffusion                   = on
Multicomponent Transport            = stefan_maxwell
Chemkin file                        = chem.bin
Surface chemkin file                = surf.bin
Transport chemkin file              = tran.bin
-----------------------------------------------------------
        Material ID Specifications
-----------------------------------------------------------
Number of Materials                 = 1
CHEMKIN                             = 0  "gaas_new"
         ELEM_BLOCK_IDS             = 1
     T_INIT                         = 500.
     U_INIT                         = 0.0
     V_INIT                         = 30 0
     W_INIT                         = 0.0
         XMF_D  AsH3       0 0044
         XMF_D GaMe3       0 00013
         XMF_D    H2       0 99547
     G_VECTOR 0.0 0.0 -980.0
END Material ID Specifications
-----------------------------------------------------------
        Boundary Condition Specifications
-----------------------------------------------------------
Number of Generalized Surfaces = 2
GENERALIZED_SURFACE 5 3
NORMAL  0.0 0 15643447 -0 98768834
TANGENT 1.0 0.0           0 0
TANGENT 0.0 0.98768834  0 15643447
GENERALIZED_SURFACE 4 3
NORMAL  0 0 0.15643447 -0 98768834
TANGENT 1.0 0.0           0.0
TANGENT 0 0 0 98768834  0.15643447
#
Number of BC                        = 33
#
```

```
# Continuation routine will overwrite the disk spin rate on the
# next 2 lines, which is currently set at 00 rpm
BC    = VEL_TAN1_BC DIRICHLET GS 1 INDEPENDENT
f_xy_spin_tilt9_disk 1
        BC_DATA =  00 0 0. 0  1.504652
BC    = VEL_TAN2_BC DIRICHLET GS 1 INDEPENDENT
f_xy_spin_tilt9_disk 1
        BC_DATA =  00.0 0. 0. 1.504652
BC    = VEL_NORM_BC DIRICHLET GS 1 DEPENDENT surface_chemkin_bc 0
BC    = VEL_TAN1_BC DIRICHLET GS 2 INDEPENDENT 0.0 0
BC    = VEL_TAN2_BC DIRICHLET GS 2 INDEPENDENT 0.0 0
BC    = VEL_NORM_BC DIRICHLET GS 2 DEPENDENT surface_chemkin_bc 0
#
BC    = T_BC DIRICHLET SS 1 INDEPENDENT 298. 0
BC    = T_BC DIRICHLET SS 4 INDEPENDENT 913  0
BC    = T_BC DIRICHLET SS 5 INDEPENDENT 913. 0
BC    = T_BC DIRICHLET SS 7 INDEPENDENT 675. 0
#
BC    = U_BC DIRICHLET SS 1 INDEPENDENT 0. 0
BC    = U_BC DIRICHLET SS 2 INDEPENDENT 0. 0
BC    = U_BC DIRICHLET SS 3 INDEPENDENT 0  0
BC    = U_BC DIRICHLET SS 4 INDEPENDENT 0. 0
BC    = U_BC DIRICHLET SS 7 INDEPENDENT 0. 0
BC    = U_BC DIRICHLET SS 8 INDEPENDENT 0. 0
BC    = U_BC DIRICHLET SS 9 INDEPENDENT 0. 0
#
BC    = V_BC DIRICHLET SS 3 INDEPENDENT 0  0
BC    = V_BC DIRICHLET SS 6 INDEPENDENT 0. 0
BC    = V_BC DIRICHLET SS 7 INDEPENDENT 0  0
BC    = V_BC DIRICHLET SS 8 INDEPENDENT 0  0
BC    = V_BC DIRICHLET SS 9 INDEPENDENT 0  0
#Set inlet flow rate here
BC    = V_BC DIRICHLET SS 1 INDEPENDENT 30 0  0
#
BC    = W_BC DIRICHLET SS 1 INDEPENDENT 0  0
BC    = W_BC DIRICHLET SS 2 INDEPENDENT 0. 0
BC    = W_BC DIRICHLET SS 3 INDEPENDENT 0  0
BC    = W_BC DIRICHLET SS 6 INDEPENDENT 0  0
BC    = W_BC DIRICHLET SS 7 INDEPENDENT 0. 0
BC    = W_BC DIRICHLET SS 8 INDEPENDENT 0  0
BC    = W_BC DIRICHLET SS 9 INDEPENDENT 0  0
#
BC    = Y_BC DIRICHLET SS 1 INDEPENDENT f_mole_fraction 1
        SPECIES_LIST = 1 2 3 4
        BC_DATA =  0 0044 0 00013 0.0 0.99547
BC    = Y_BC NEUMANN   SS 5 DEPENDENT surface_chemkin_bc 1
        SPECIES_LIST = ALL
        SURF_SPECIES_LIST = GaMe(S) AsH2(S) BLOCK Ga-GaAs(D) As-
GaAs(D)
        BC_DATA = FLOAT 0.2 0.4 0.4 1.0 1.0
BC    = Y_BC NEUMANN   SS 4 DEPENDENT surface_chemkin_bc 3
        SPECIES_LIST = ALL
        SURF_SPECIES_LIST = GaMe(S) AsH2(S) BLOCK
        BC_DATA = FLOAT 0.2 0.4 0.4
        SURF_SPECIES_LIST =  Ga-GaAs(D)
        BC_DATA = FLOAT 1.0
        SURF_SPECIES_LIST = As-GaAs(D)
        BC_DATA = FLOAT 1.0
-----------------------------------------------------------
        Initial Guess/Condition Specifications
-----------------------------------------------------------
Set Initial Condition/Guess                  = constant 0.0
Apply function                               = no
Time Index to Restart From                   = 1
-----------------------------------------------------------
                  Output Specifications
-----------------------------------------------------------
User Defined Output                                    = yes
Parallel Output                                        = no
Scalar Output                                          = yes
Time Index to Output To                                = 1
Nodal variable output times
        every 1 steps

Number of nodal output variables                       = 4
Nodal variable names:
        Temperature
        Velocity
        Pressure
        Mass_fraction
-----------------------------------------------------------
      Data Specification for User's Functions
-----------------------------------------------------------
Number of functions to pass data to    =  1
Function Name      =    f_xy_spin_average 2
#
FN_DATA = INT 5 5
FN_DATA = FLOAT 0  0  1.504652
```

*Figure D.9. MPSalsa input file for the* Tilted Reactor *example problem.*

A typical solution is shown in Figure D.10, which includes the streamlines through the domain and the contours of the reactant (GaMe3) on the surface. The effect of the counter-clockwise rotating disk on the flow and surface concentrations can be seen.

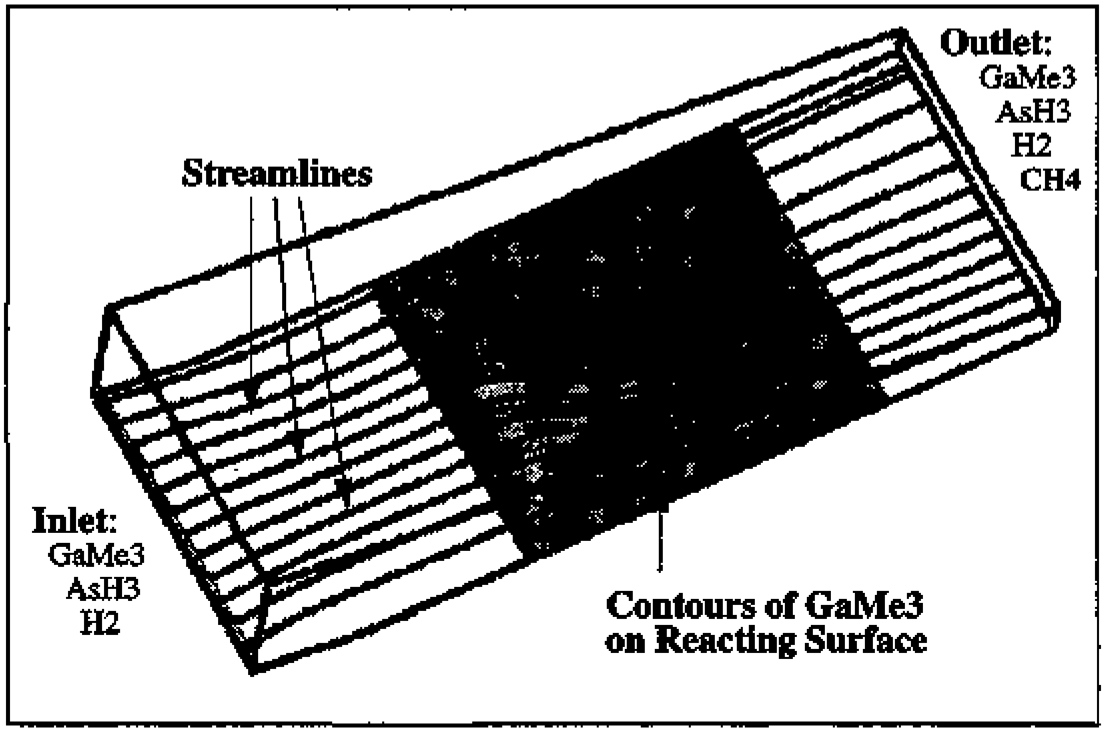


*Figure D.10. Streamlines and surface concentrations for a solution to the* Tilted Reactor *example problem.*

Figure D.11 shows the time-averaged (spin-averaged) deposition profiles over the disk for the four different spin rates calculated in the one continuation run. (The profiles are calculated using a non-standard post-processing routine, `f_xy_spin_average`, which expands the radial variation in the deposition as a series of orthonormal polynomials.) The disk rotation rate is seen to be a minor factor in the non-uniformity of the deposition, but it can be seen that rotation degrades uniformity.

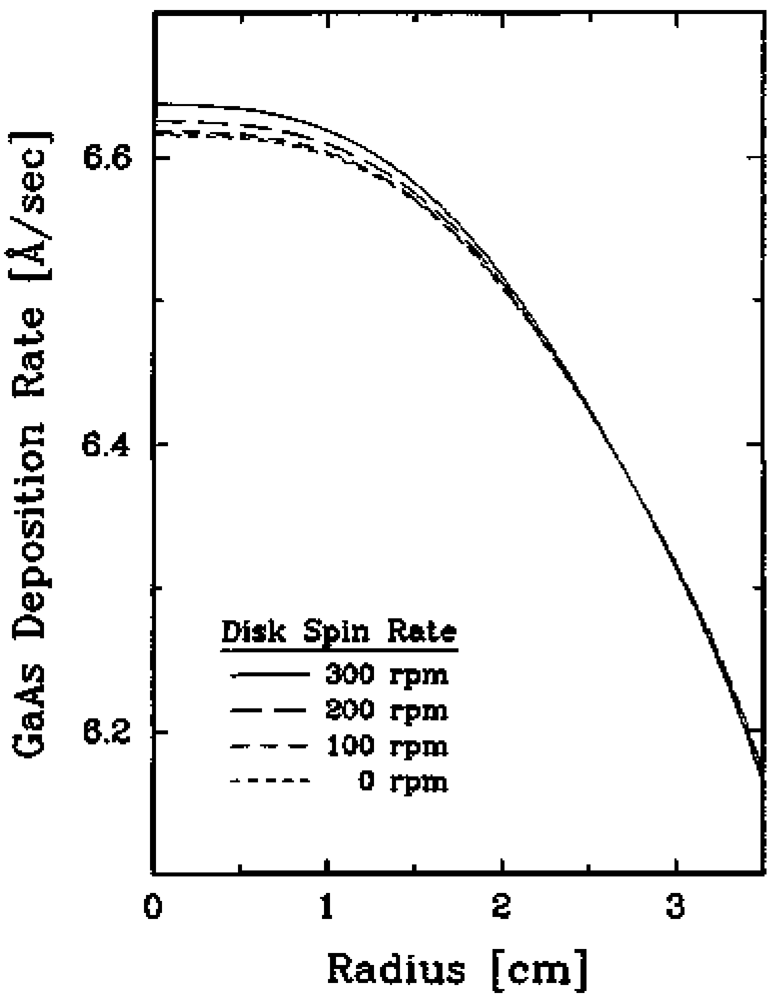


*Figure D.11. Plot of the spin-averaged deposition rate on the rotating disk in the* Tilted Reactor *example problem for the 4 different spin rates.*

## References

1. T.D. Blacker. "FASTQ Users Manual, Version 2.1," Sandia National Laboratories Tech. Rep. SAND88-1326, Albuquerque, NM (1988).
2. W.G. Breiland and G.H. Evans. "Design and Verification of Nearly Ideal Flow and Heat Transfer in a Rotating Disk CVD Reactor," *J. Electrochem Soc*, **138**(6) (1991).
3. A.N. Brooks and T.J.R. Hughes. "Strealmine Upwind/Petrov-Galerkin Formulations for Convection Dominated Flows with Particular Emphasis on the Incompressible Navier-Stokes Equations," *Computer Methods in Applied Mechanics and Eng.*, **32** (1982) 199–259.
4. S. Carney, M. Heroux and G. Li. "A proposal for a sparse BLAS toolkit," SPARKER Working Note #2, Cray Research, Inc., Eagen, MN (1993).
5. M.E. Coltrin, R.J. Kee, F.M. Rupley, and E. Meeks. "Surface Chemkin-III: A FORTRAN package for analyzing heterogeneous chemical kinetics at a solid-surface-gas-phase interface," Sandia National Laboratories Tech. Rep. SAND96-8217, Albuquerque, NM (1996).
6. M.E. Coltrin, R.J. Kee, G.H. Evans, E. Meeks, F.M. Rupley, and J.F. Grcar. "SPIN: A Fortran Program for Modeling One-Dimensional Rotating-Disk/Stagnation-Flow Chemical Vapor Deposition Reactors," Sandia National Laboratories Tech. Rep. SAND87–8248, Albuquerque, NM (1987).
7. M.E. Coltrin and H.K. Moffat. "Surftherm: A Program to Analyze Thermochemical and Kinetic Data in Gas-Phase and Surface Chemical Reaction Mechanisms," Sandia National Laboratories Tech. Rep. SAND94–0219, Albuquerque, NM (1996).
8. M.S. Eldred, W.E. Hart, W.J. Bohnhoff, V.J. Romero, S.A. Hutchinson, and A.G. Salinger. "Utilizing Object-Oriented Design to Build Advanced Optimization Strategies with Generic Implementation," *Proceedings of the 6th AIAA/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, AIAA-96-4164-CP, Bellevue, WA, (1996) 1568-1582.
9. S.C. Eisenstat and H.F. Walker. "Choosing the forcing terms in an inexact Newton method," *SIAM J. Sci. Comput.*, **17** (1996) 16-32.
10. S.C. Eisenstat and H.F. Walker. "Globally convergent inexact Newton methods," *SIAM j. Optimization*, **4** (1994) 393-422.
11. C.R. Ethier and D.A. Steinman. "Exact fully 3D Navier-Stokes solutions for benchmarking," *Int J. Num. Meth. Fluids*, **19** (1994) 369-375.
12. G. Evans and R. Greif. "A Numerical Model of the Flow and Heat Transfer in a Rotating Disk CVD Reactor," *J. Heat Transfer*, **109** (197).
13. FIDAP 7.0 Theory Manual. Fluid Dynamic International, Inc. (1984) Chapter 6, 14-15.
14. W.C. Gardiner and and J. Troe. "Rate coefficients of thermal dissociation, isomerization and recombination reactions," in *Combustion Chemistry*, Ed. W.C. Gardiner, Springer-Verlag, New York (1984).

15. D.K. Gartling. "Merlin II - A computer program to transfer solution data between finite element meshes," Sandia National Laboratories Tech. Rep. SAND89-2989, Albuquerque, NM, (1991).

16. D.K. Gartling and R.E. Hogan. "Coyote II -- A finite element computer program for nonlinear heat conduction problems, Part 1 -- Theoretical development," Sandia National Laboratories Tech. Rep. SAND94-1173, Albuquerque, NM (1994).

17. A.P. Gilkey and G.D. Sjaardema. "GEN3D: A GENESIS Database 2D to 3D Transformation Program," Sandia National Laboratories Tech. Rep. SAND89-0485, Albuquerque, NM (1989).

18. G.H. Golub and C.F. Van Loan. *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD (1983) 150-153.

19. P.M. Gresho, S.T. Chan, R.L. Lee, and C.D. Upson. "A modified finite element method for solving the time-dependent, incompressible Navier-Stokes equations: part 2: applications," *Int J Numer Meth Fluids*, **4** (1984) 619-640.

20. P.M. Gresho, R.L. Lee, and R.L. Sani. "On the time-dependent solution of the incompressible Navier-Stokes equations in two and three dimensions," in *Recent Advances in Numerical Methods in Fluids*, C. Taylor and K. Morgan, eds., Pineridge Press Ltd., Swansea, UK (1980) 27-81.

21. B. Hendrickson and R. Leland. "An improved spectral graph partitioning algorithm for mapping parallel computations," Sandia National Laboratories Tech. Rep. SAND92–1460, Sandia National Laboratories, Albuquerque, NM (1992).

22. B. Hendrickson and R. Leland. "The Chaco User's Guide, Version 2.0," Sandia National Laboratories Tech. Rep. SAND94-2692, Albuquerque, NM (1995).

23. G.L. Hennigan and J.N. Shadid. "NemesisI : A set of functions for describing unstructured finite-element data on parallel computers," Sandia National Laboratories Tech. Rep. in preparation, Albuquerque, NM.

24. J.R. Hipp, R.R. Lober, S.A. Mitchell, G.D. Sjaardema, M.K. Smith, T.J. Tautges, T.J. Wilson, W.R. Oakes, et al. "CUBIT Mesh Generation Environment Volume 1: User's Manual," Sandia National Laboratories Tech. Rep. SAND94–1100, Albuquerque, NM (1996).

25. T.J.R. Hughes, L.P. Franca, and M. Balestra. "A New Finite Element Formulation for Computational Fluid Dynamics: V. Circumventing the Babuska-Brezzi Condition: A Stable Petrov-Galerkin Formulation of the Stokes Problem Accommodating Equal-order Interpolations," *Computer Methods in Applied Mechanics and Eng.*, **59** (1986) 85–99.

26. S.A. Hutchinson, J.N. Shadid, and R.S. Tuminaro. "Aztec User's Guide: Version 1.0," Sandia National Laboratories Tech. Rep. SAND95–1559, Albuquerque, NM (1995).

27. R.J. Kee, G. Dixon-Lewis, J. Warnatz, M.E. Coltrin, and J.A. Miller. "A FORTRAN Computer Code Package for the Evaluation of Gas-Phase, Multicomponent Transport Properties," Sandia National Laboratories Tech. Rep. SAND86–8246, Albuquerque, NM (1986).

28. R.J. Kee, F.M. Rupley, E. Meeks, and J.A. Miller. "Chemkin-III: A Fortran Chemical Kinetics Package for the Analysis of Gas-Phase Chemical Kinetics," Sandia National Laboratories Tech. Rep. SAND96–8215, Albuquerque, NM (1996).

29. B. Kernighan and S. Lin. "An efficient heuristic procedure for partitioning graphs," *Bell System Technical Journal,* **29** (1970) 291–307.

30. K.J. Laidler. *Chemical Kinetics*, Harper & Row, New York (1987).

31. *LAPACK User's Guide*, http://www.netlib.org/lapack/lug/lapack_lug.html

32. M.J. Martinez and P.L. Hopkins, private communication.

33. E. Meeks, H.K. Moffat, J.F. Grcar, and R.J. Kee. "AURORA: A FORTRAN Program for Modeling Well Stirred Plasma and Thermal Reactors with Gas and Surface Reactions," Sandia National Laboratories Tech. Rep. SAND96–8218, Albuquerque, NM (1996).

34. Message Passing Interface Forum. "MPI: A Message-Passing Interface Standard," University of Tennessee, Knoxville, TN (1995).

35. H.K. Moffat, K.P. Killeen, and K.C. Baucom. "Group V Inhibition of GaAs and AlAs MOCVD Growth Rates," submitted (1995).

36. S.V. Patanker. *Numerical heat tranfer and fluid flow,* Hemisphere Publishing Corp., London (1980).

37. R. Rew, G. Davis, and S. Emerson, "NetCDF User's Guide: an interface for data access, version 2.3," UCAR (1993).

38. A.G. Salinger, S. Brandon, R. Aris, and J.J. Derby. "Buoyancy-Driven Flows of a Radiatively Particilating Fluid in a Vertical Cylinder Heated from Below," *Proc Royal Soc London* A **442** (1993).

39. H. Schlichting. *Boundary Layer Theory,* 7th Ed., McGraw-Hill, New York (1979).

40. L.A. Schoof and V.R. Yarberry. "ExodusII: A Finite Eelement Data Model," Sandia National Laboratories Tech. Rep. SAND94-2137, Albuquerque, NM, (1994).

41. J.N. Shadid. "Experimental and Computational Study of the Stability of Natural Convection Flow in an Inclined Enclosure," Ph.D. Dissertation, University of Minnesota (1989).

42. J.N. Shadid, H.K. Moffat, S.A. Hutchinson, G.L. Hennigan, K.D. Devine, and A.G. Salinger. "MPSalsa: A Finite Element Computer Program for Reacting Flow Problems, Part 1 – Theoretical Development," Sandia National Laboratories Tech. Rep. SAND95–2752, Albuquerque, NM (1996).

43. J.N. Shadid and R.S. Tuminaro. "Sparse iterative algorithm software for large-scale MIMD machines: an initial discussion and implementation," *Concurrency: Practice and Experience,* **4** (1992) 481–497.

44. J.N. Shadid and R.S. Tuminaro. "A comparison of preconditioned nonsymmetric Krylov methods on a large-scale MIMD machine," *SIAM J. Sci. Stat. Comput.*, **15** (1994) 440-459.

45. R.K. Shah and A.L. London. *Laminar Flow Forced Convection in Ducts*, Academic Press, New York (1978).

46. R.K. Shah and M.S. Bhatti. "Laminar Convective Heat Transfer in Ducts," in *Handbook of Single-Phase Convective Heat Transfer*, S. Kakac, R.K. Shah, and W. Aung (eds.), Wiley & Sons (1987).

47. D.C. Sorensen and R.B. Lehoucq, Department of Computational and Applied Mathematics, Rice University, Houston, Texas.

48. T.E. Tezduyar, S. Mittal, S.E. Ray, and R. Shih. "Incompressible Flow Computations with Stabilized Bilinear and Linear Equal-order-interpolation Velocity-Pressure Elements," *Computer Methods in Appl. Mechanics and Eng.*, **95** (1992) 221–242.

49. Z. Zlatev, V.A. Barker, and P.G. Thomsen. "SSLEST--a FORTRAN IV subroutine for solving sparse systems of linear equations: User's guide," Technical Report, Institute for Numerical Analysis, Technical University of Denmark, Lyngby, Denmark (1978).

## Index

## Distribution

EXTERNAL DISTRIBUTION:

Steve Ashby
Lawrence Livermore Nat. Lab.
M/S L-316
PO Box 808
Livermore, CA 94551-0808

Rob Bisseling
Department of Mathematics
Budapestlaan 6, De Uithof, Utrecht
PO Box 80.010, 3508 TA Utrecht
The Netherlands

Petter Bjorstad
University of Bergen
Institutt for Informatikk
Thomohlengst 55
N-5008 Bergen, Norway

Randall Bramley
Dept. of CSci.
Indiana University
Bloomington IN 47405

Rich A. Cairncross
Mechanical Engineering Department
University of Delaware
313 Spencer Laboratory
Newark, DE 19716-3140

G. F. Carey
ASE/EM Dept., WRW 305
University of Texas
Austin, TX, 78712

Steven P. Castillo
Klipsch School of Electrical & Computer Eng.
New Mexico State University
Box 30001
Las Cruces, NM 88003-0001

J. M. Cavallini
US Department of Energy
OSC, ER-30, GTN
Washington, DC 20585

T. Chan
UCLA
405 Hilgard Ave.
Los Angeles, CA 90024-7009

Warren Chernock
Scientific Advisor DP-1
US Department of Energy
Forestal Bldg. 4A-045
Washington, DC 20585

Doug Cline
The University of Texas System
Center for High Performance Computing
%Balcones Research Center
10100 Burnett Road, CMS 1.154
Austin, Texas 78758

Vernon Cole
Equipment Simulation Group, APRDL
3501 Ed Bluestein Boulevard, MD: K-10
Austin, TX 78721

Tom Coleman
Dept. of Computer Science
Upson Hall
Cornell University
Ithaca, NY 14853

Prof. D. S. Dandy
Colorado State Univ.
Dept. Agriculture and Chem. Eng.
Fort Collins, CO 80523

Prof. J. J. Derby
Dept. of Chemical Eng. and Materials Science
University of Minnesota
421 Washington Ave. S.E.
Minneapolis, MN 55455

J. J. Dongarra
Computer Science Dept.
104 Ayres Hall
University of Tennessee
Knoxville, TN 37996-1301

I. S. Duff
CSS Division
Harwell Laboratory
Oxfordshire, OX11 ORA
United Kingdom

Erik Egan
Equipment Simulation Group, APRDL
3501 Ed Bluestein Boulevard, MD: K-10
Austin, TX 78721

Alan Edelman
Dept. of Mathematics
MIT
Cambridge, MA 02139
%edelman@math.mit.edu

Steve Elbert
US Department of Energy
OSC, ER-30, GTN
Washington, DC 20585

H. Elman
Computer Science Dept.
University of Maryland
College Park, MD 20842

R. E. Ewing
Mathematics Dept.
University of Wyoming
PO Box 3036 University Station
Laramie, WY 82071

Charbel Farhat
Dept. Aerospace Engineering
UC Boulder
Boulder, CO 80309--0429

J. E. Flaherty
Computer Science Dept.
Rensselaer Polytechnic Inst.
Troy, NY 12180

G. C. Fox
Northeast Parallel Archit. Cntr.
111 College Place
Syracuse, NY 13244

R. F. Freund
NRaD- Code 423
San Diego, CA 99152-5000

D. B. Gannon
Computer Science Dept.
Indiana University
Bloomington, IN 47401

Horst Gietl
nCUBE Deutschland
Hanauer Str. 85
8000 Munchen 50
Germany

Paul Giguere
Group TSA-8
MS K575
Los Alamos National Laboratory
Los Alamos, NM 87545

John Gilbert
Xerox PARC
3333 Coyote Hill Road
Palo Alto, CA 94304

R. J. Goldstein
Mechanical Engineering Department
University of Minnesota
111 Church St.
Minneapolis, MN 55455

G. H. Golub
Computer Science Dept.
Stanford University
Stanford, CA 94305

Anne Greenbaum
New York University
Courant Institute
251 Mercer Street
New York, NY 10012-1185

Satya Gupta
Intel SSD
Bldg. CO6-09, Zone 8
14924 NW Greenbrier Parkway
Beaverton, OR, 97006

J. Gustafson
Computer Science Dept.
236 Wilhelm Hall
Iowa State University
Ames, IA 50011

Doug Harless
NCUBE
2221 East Lamar Blvd., Suite 360
Arlington, TX 76006

Michael Heath
Univ. of Ill., Nat. CSA
4157 Bechman Institute
405 North Matthews Ave.
Urbana, IL 61801-2300

Mike Heroux
Cray Research Park
655F Lone Oak Drive
Eagan, MN 55121

Dan Hitchcock
US Department of Energy
SCS, ER-30 GTN
Washington, DC 20585

Fred Howes
US Department of Energy
OSC, ER-30, GTN
Washington, DC 20585

Prof. Marylin C. Huff
Department of Chemical Engineering
University of Delaware
Newark, DE 19716

Prof. Michael K. Jensen
Rensselaer Polytechnic Institute
Troy, NY 12180-3590

Prof. K. J. Jensen
Massachusetts Institute of Technology
Dept. Chem. Eng. MIT 66-566
Cambridge, Mass. 02139-4307

Christopher R. Johnson
Department of Computer Science
3484 MEB
University of Utah
Salt Lake City, UT 84112

David Keyes
NASA Langley Research Center
ICASE
M/S 132C
Hampton, VA 23681-0001

David Kincaid
Center for Numerical Analysis
RLM 13.150
University of Texas
Austin, TX 78713-8510

T. A. Kitchens
US Department of Energy
OSC, ER-30, GTN
Washington, DC 20585

Vipin Kumar
Computer Science Department
Institute of Technology
200 Union Street S.E.
Minneapolis, MN 55455

Joanna Lees
Intel Corp.
Scalable Systems Division
CO1-15
15201 NW Greenbrier Parkway
Beaverton, OR 97006

John Lewis
Boeing Corp.
M/S 7L-21
P.O. box 24346
Seattle, WA 98124-0346

T. A. Manteuffel
Department of Mathematics
University of Co. at Denver
Denver, CO 80202

S. F. McCormick
Univ. of Colorado
Program in Applied Mathematics
Campus Box 526
Boulder, CO 80309-0526

Computer Mathematics Group
University of CO at Denver
1200 Larimer St.
Denver, CO 80204

Robert McLay
University of Texas at Austin
Dept. ASE-EM
Austin, TX 78712
%mclay@cfdlab.ae.utexas.edu

P. C. Messina
158-79
Mathematics & Comp. Sci. Dept.
Caltech
Pasadena, CA 91125

C. Moler
The Mathworks
24 Prime Park Way
Natick, MA 01760

Gary Montry
Southwest Software
11812 Persimmon, NE
Albuquerque, NM 87111

D. B. Nelson
US Department of Energy
OSC, ER-30, GTN
Washington, DC 20585

Kwong T. Ng
Klipsch School of Electrical & Computer Eng.
New Mexico State University
Box 30001
Las Cruces, NM 88003-0001

S. V. Patankar
Mechanical Engineering Department
University of Minnesota
111 Church St.
Minneapolis, MN 55455

Linda Petzold
L-316
Lawrence Livermore Natl. Lab.
Livermore, CA 94550

Barry Peyton
Mathematical Sciences Section
Oak Ridge National Laboratory
PO. Box 2008, Bldg. 6012
Oak Ridge, TN 37831-6367

Paul Plassman
Math and Computer Science Division
Argonne National Lab
Argonne, IL 60439

Claude Pommerell
AT&T Bell Labs
600 Mountain Ave., Room 2C-548A
Murray Hill, NJ 07974--0636

Alex Pothen
Department of Computer Science
Old Dominion University
Norfolk, VA 23529-0162

J. Rattner
Intel Scientific Computers
15201 NW Greenbriar Pkwy.
Beaverton, OR 97006

Patrick Riley
Intel-SSD
600 S. Cherry St., Suite 700
Denver, CO 80222

Ed Rothberg
Silicon Graphics, Inc.
MS 7L-580
2011 N. Shoreline Blvd.
Mountain View, CA 94043

Y. Saad
University of Minnesota
4-192 EE/CSci Bldg.
200 Union St.
Minneapolis, MN 55455-0159

Joel Saltz
Computer Science Department
A.V. Williams Building
University of Maryland
College Park, MD 20742

A. H. Sameh
CSRD, University of Illinois
305 Talbot Laboratory
104 S. Wright St.
Urbana, IL 61801

P. E. Saylor
Dept. of Comp. Science
222 Digital Computation Lab
University of Illinois
Urbana, IL 61801

Carl Scarbnick
San Diego Supercomputer Center
P.O. Box 85608
San Diego, CA 92186-9784

Rob Schreiber
RIACS
NASA Ames Research Center
Mail Stop T045-1
Moffett Field, CA 94035-1000

M. H. Schultz
Department of Computer Science
Yale University
PO Box 2158
New Haven, CT 06520

Mark Seager
LLNL, L-80
PO box 803
Livermore, CA 94550

T. W. Simon
Mechanical Engineering Department
University of Minnesota
111 Church St.
Minneapolis, MN 55455

Richard Sincovec
Mathematical Sciences Section
Oak Ridge Nat. Lab.
P.O. Box 2008, Bldg. 6012
Oak Ridge, TN 37831-6367

Vineet Singh
HP Labs, Bldg. 1U, MS 14
1501 Page Mill Road
Palo Alto, CA 94304

Anthony Skjellum
Mississippi State University
Computer Science
PO Drawer CS
Mississippi State, MS 39762

L. Smarr
Director, Supercomputer Apps.
152 Supercomputer Applications
Bldg. 605 E. Springfield
Champaign, IL 61801

Burton Smith
Tera Computer Co
400 N. 34th St., Suite 300
Seattle, WA 98103

Harold Trease
Los Alamos National Lab
PO Box 1666, MS F663
Los Alamos, NM 87545

C. VanLoan
Department of Computer Science
Cornell University, Rm. 5146
Ithaca, NY 14853

John VanRosendale
ICASE, NASA Langley Research Center
MS 132C
Hampton, VA 23665

Steve Vavasis
Department of Computer Science / ACRI
722 Engineering and Theory Center
Cornell University
Ithaca, NY 14853

R. G. Voigt
MS 132-C
NASA Langley Resch Cntr, ICASE
Hampton, VA 36665

Phuong Vu
Cray Research, Inc.
19607 Franz Road
Houston, TX 77084

Steven J. Wallach
Convex Computer Corp.
3000 Waterview Parkway
PO Box 833851
Richardson, TX 75083-3851

G. W. Weigand
U.S. DOE
1000 Independence Ave., SW
Room 4A-043 (DP1.1)
Washington, DC 20585

Olof B. Widlund
Dept. Computer Science
Courant Inst., NYU
251 Mercer St.
New York, NY 10012

INTERNAL DISTRIBUTION:

| | | |
|---|---|---|
| 1 | MS 0151 | Gerold Yonas, 9000 |
| 1 | MS 0321 | William Camp, 9200 |
| 1 | MS1427 | P. Mattern, 1100 |
| 1 | MS0601 | P. Esherick, 1126 |
| 10 | MS 0601 | Harry K. Moffat, 1126 |
| 1 | MS0601 | M. E. Coltrin, 1126 |
| 1 | MS 0827 | J. S. Rottler, 5600 |
| 1 | MS 1111 | Sudip Dosanjh, 9221 |
| 10 | MS 1111 | Scott Hutchinson, 9221 |
| 30 | MS 1111 | John N. Shadid, 9221 |
| 30 | MS 1111 | Andrew G. Salinger, 9221 |
| 10 | MS 1111 | Gary L. Hennigan, 9221 |
| 10 | MS 1111 | Rod C. Schmidt 9221 |
| 1 | MS 1111 | Daniel Barnette, 9221 |
| 1 | MS 1111 | Steven J. Plimpton, 9221 |
| 1 | MS 1111 | David R. Gardner, 9221 |
| 1 | MS 1111 | Matt St. John, 9921 |
| 1 | MS 1110 | Richard C. Allen, 9222 |
| 1 | MS 1110 | David E. Womble, 9222 |
| 1 | MS 1110 | Ray S. Tuminaro, 9222 |
| 1 | MS 1110 | Lydie Prevost, 9222 |
| 1 | MS 1109 | Art Hale, 9224 |
| 1 | MS 1109 | Ted Barragy, 9224 |
| 1 | MS 1109 | Bob Benner, 9224 |
| 1 | MS 1109 | James Tomkins, 9224 |
| 1 | MS 1111 | Mark P. Sears, 9225 |
| 10 | MS 1111 | Karen Devine, 9226 |
| 1 | MS 1111 | Robert W. Leland, 9226 |
| 1 | MS 1111 | Bruce A. Hendrickson, 9226 |
| 1 | MS 1111 | Courtenay Vaughn, 9226 |
| 1 | MS 0441 | S. W. Attawy, 9226 |
| 1 | MS 0441 | L. A. Schoof, 9215 |
| 1 | MS 0441 | T. J. Tauges, 9226 |
| 1 | MS 0819 | J. Michael McGlaun, 9231 |
| 1 | MS 0819 | James S. Perry, 9231 |
| 1 | MS 0819 | Allem C. Robinson, 9231 |
| 1 | MS 0439 | David R. Martinez, 9234 |
| 1 | MS 0841 | P. L. Hommert, 9100 |
| 1 | MS 0833 | Johnny H. Biffle, 9103 |
| 1 | MS 0841 | E. D. Gorham, 9104 |
| 1 | MS 0843 | A. C. Ratzel, 9112 |
| 1 | MS 0834 | M. R. Baer, 9112 |
| 1 | MS 0834 | A. S. Geller, 9112 |
| 1 | MS 0834 | R. R. Torczynski, 9112 |

| | | |
|---|---|---|
| 1 | MS 0826 | W. L. Hermina, 1553 |
| 1 | MS 0826 | T. J. Bartel, 9153 |
| 1 | MS 0825 | C. C. Wong, 9154 |
| 1 | MS 0825 | Basil Hassan, 9155 |
| 1 | MS 0437 | G. D. Sjaardema |
| 1 | MS 0827 | Dave K. Gartling, 9111 |
| 1 | MS 0827 | Randy Schunk, 9111 |
| 1 | MS 0827 | Phil Sackinger, 9111 |
| 1 | MS 0827 | Mario Martinez, 9111 |
| 1 | MS 0827 | Mike Glass, 9111 |
| 1 | MS 0827 | Bob McGrath, 9111 |
| 1 | MS 0827 | Polly Hopkins, 9111 |
| 1 | MS 0827 | Jim Schutt, 9111 |
| 1 | MS 0827 | Melinda Sirman, 9111 |
| 1 | MS 0827 | Steve Kempka, 9111 |
| 1 | MS 0834 | Robert B. Campbell, 9112 |
| 1 | MS 0835 | Roy E. Hogan Jr., 9111 |
| 1 | MS 0835 | Mark A. Christon, 9111 |
| 1 | MS 0826 | Robert J. Cochran, 9114 |
| 1 | MS 0750 | Greg A. Newman, 6116 |
| 1 | MS 0750 | David L. Alumbaugh, 6116 |
| 1 | MS 9214 | Juan Meza, 8117 |
| 1 | MS 9042 | Joseph F. Grcar, 8745 |
| 1 | MS 9042 | Chris Moen, 8745 |
| 1 | MS 9042 | Fran Rupley, 8745 |
| 1 | MS 9042 | S. K. Griffiths, 8745 |
| 1 | MS 9042 | Greg Evans, 8745 |
| 1 | MS 9051 | W. T. Ashurst, 8351 |
| 1 | MS 9051 | Alan Kerstein, 8351 |
| 1 | MS 9051 | Jackie Chen, 8351 |
| 1 | MS 9051 | H. Najm, 8351 |
| 1 | MS 9018 | Central Technical Files, 8523-2 |
| 5 | MS 0899 | Technical Library, 4414 |
| 2 | MS 0619 | Review & Approval Desk, 12630 For DOE/OSTI |