INL/EXT-05-00159

# Programmers Manual For The PVMEXEC Program

Walter L. Weaver III

March 2005


The INL is a U.S. Department of Energy National Laboratory
operated by Battelle Energy Alliance

INL/EXT-05-00159

# Programmers Manual for the PVMEXEC Program

**Walter L. Weaver III**

**March 2005**

**Idaho National Laboratory**
**Idaho Falls, Idaho 83415**

**Prepared Under DOE/NE Idaho Operations Office**
**Contract No. DE-AC07-05ID14517**

# ABSTRACT

This report describes the implementation of the PVMEXEC program. The information in the report is intended for programmers wanting to modify the PVMEXEC program.

# CONTENTS

# 1 Introduction

The purpose of this report is to document the implementation of the PVMEXEC computer program. The responsibilities of the PVMEXEC program were documented implicitly in the description of the PVMEXEC Application Coupling Interface (API)[1]. That document describes the modifications that must be implemented in a computer code to allow it to participate in a coupled simulation. Implicit within the requirements for the coupled codes are the requirements for the PVMEXEC program that initiates, controls, and terminates coupled computations. This report first describes the responsibilities of the PVMEXEC program. It then describes how these responsibilities are implemented in the PVMEXEC program and its computational procedures. This information can be used by programmers to understand how the program works and how it can be modified to correct problems or to implement new capabilities.

## 1.1 Background

The PVMEXEC program was developed to facilitate the simulation of a system (e.g., a nuclear power plant) using several different computer programs to describe the transient behavior of the system. Simulation codes are generally written to provide detailed models of some portion of a system, i.e., COBRA[2], RELAP5-3D[3], TRAC-PF1/MOD1[4], FLUENT[5] and TRACE[6] for the fluid systems, CONTAIN[7] and MELCOR[8] for the containment systems, NESTLE[9] and PARCS[10] for reactor power, etc. The PVMEXEC program and the code coupling system that it implements enables the use of different codes for the simulation of different portions of the system in a unified analysis of the transient behavior of the system. Each code in the coupled simulation needs data from some of the others codes in the simulation. The data is passed between the several codes using the Parallel Virtual Machine[11] (PVM) message passing methodology. The PVMEXEC program was originally developed to couple the RELAP5-3D code to other thermal-hydraulic codes, however any code that implements the PVMEXEC API may be used in a coupled simulation.

The material in this report describes the use of the procedures (i.e., subroutines and functions) in the PVM software library to implement the functionality of the PVMEXEC program. This document is not a tutorial on the use of PVM and assumes that the reader is familiar with PVM. There is an old adage that ‘the code is its own best documentation’. Even though that is true, it is often difficult to ‘reverse engineer’ the coding to understand what the programmer was trying to accomplish with a particular section of coding, particularly if there is an error in the code. This document explains WHAT each section of coding is intended to accomplish but not necessarily explain the details of HOW it is accomplished. It is assumed that the reader of this document has access to the program source code for reference as the reader reads this document. The Fortran interface to the PVM library modules (the underlying modules are written in C) is described because the PVMEXEC program is written in Fortran 90 (F90), although only a few features of F90 are used by the PVMEXEC program. In some cases, the Fortran interface to the underlying C modules provided by the PVM library could not be used and wrapper functions were written for the underlying C functions (e.g., if the Fortran interface to the underlying C function in the PVM library omitted some parameters from the Fortran interface that are available in the C interface).

## 1.2 Responsibilities of the PVM Program

The coupling system that the PVMEXEC program implements was designed as a batch system. The PVMEXEC program is executed by the user and the simulation is expected to proceed to the end without user interaction with the PVMEXEC program. However, user interaction with the programs executing in the coupled computation is not precluded. The PVMEXEC program and the several simulation codes are expected to read their input files and behave according to the specifications in the input files. The PVMEXEC program has several responsibilities. These responsibilities roughly correspond to the several sections of its input file. These responsibilities include: 1) configure the virtual machine, 2) execute the several simulation codes on the virtual machine, 3) tell each of the simulation codes what data it is to exchange with the other simulation codes, and 4) manage the time advancements of the simulation codes participating in the coupled simulation including the production of printed or plottable output. An overall responsibility of the PVMEXEC program is to terminate the coupled computation gracefully if any of the simulation codes terminate unexpectedly or encounter problems from which they cannot recover. The PVMEXEC program writes information to its output file. It also writes the status information to the terminal screen so that the user may monitor the progress of the simulation.

## 1.3 Structure of PVMEXEC

The PVMEXEC program consists of a number of procedures (programming units). These procedures are the main program, three principal subroutines, and a number of auxiliary subroutines and functions. These procedures make extensive use of the subroutines and functions contained in the PVM software library. Section 2 contains a detailed description of the main program and the principle subroutines. Section 3 contains a list of all of the procedures along with a short description of their purpose. The following sections contain a more complete description of the main program and the three principal subroutines.

### 1.3.1 Program PVMEXEC

Program PVMEXEC is the main program. It performs a number of functions including configuration and startup of the virtual machine, execution of the several simulation codes on the virtual machine, specification of the data that is to be exchanged between the several simulation codes (subroutine INPUTD), coordination of the initial data exchange between the several simulation codes, execution of the transient simulation (subroutine TRAN), termination of the several simulation codes if there are unrecoverable errors during any phase of the coupled computation, and shutdown of the virtual machine when the coupled simulation has finished, either because the simulation has reached its ending time or because of an unrecoverable error in a simulation code.

### 1.3.2 Subroutine INPUTD

Subroutine INPUTD reads the input file of the PVMEXEC program, configures and starts the virtual machine, and tells each simulation code what data it is to exchange with the other codes participating in the coupled simulation.

### 1.3.3 Subroutine TRAN

Subroutine TRAN controls the time step advancements for a coupled computation. It periodically writes information to the output file of the PVMEXEC program and to the terminal screen to advise the user of the progress and status of the coupled computation. It detects time step failure and coordinates time step backup and repeats for semi-implicit thermal-hydraulic coupling. Finally, it calls subroutine DTSTEP for the computation of time step size for synchronous coupling, etc.

### 1.3.4 Subroutine DTSTEP

Subroutine DTSTEP computes the global time step size for synchronous explicit thermal-hydraulic coupling, for semi-implicit thermal-hydraulic coupling, and for kinetics and control systems coupling. It computes the global explicit coupling interval for asynchronous explicit thermal-hydraulic coupling. It controls the production of output by the several simulation codes - printable and plottable output as well as restart information. Finally, it coordinates the data exchanges between codes participating in explicit thermal-hydraulic coupling.

# 2 Principal Programming Units

The following sections contain a detailed description of the principal programming units that comprise the PVMEXEC program. These procedures encapsulate the main functionality of the PVMEXEC program. They use other auxiliary procedures during their execution to perform their assigned tasks. All of the programming units in the PVMEXEC program will be described in the next section of this report. This section only describes the principal programming units of the PVMEXEC program so that the reader might understand how the program works without being overwhelmed by details. The order of the descriptions is the order in which the procedures (subroutines and functions) are first called during a coupled simulation. A subroutine or function may be called many times during the execution of a coupled transient simulation but it is only described here once.

## 2.1 Program PVMEXEC

Program PVMEXEC is the main program for the PVMEXEC program. The order of computations is as follows.

1. Call subroutine GNINIT1. Subroutine GNINIT1 reads the command line parameters and sets file names for the input and output files. It opens the input file for reading and the output file for writing.

2. Call F90 data and time routine to establish the date and start time of the coupled simulation. Write the date and start time to the output file and the terminal screen.

3. Initialize timer by a call to subroutine GETSEC. The times return by calls to GETSEC are used to determine when to write the status information to the output file and the terminal screen.

4. Call subroutine INPUTD. Subroutine INPUTD reads the input file, generates a database of information about the virtual machine and the instances of the simulation codes. It starts up the virtual machine, executes the several instances of the simulation codes on the virtual machine, and specifies to each of the simulation codes the data that it is to exchange with the other codes in the coupled simulation.

5. Check for an error condition after the return from subroutine INPUTD. If there is an error condition, terminate the coupled simulation.

6. If there are no errors, coordinate the data exchanges between the codes participating in explicit thermal-hydraulic coupling for each type of explicit thermal-hydraulic coupling, first for synchronous parallel explicit coupling, next for asynchronous parallel explicit coupling, then for synchronous sequential explicit coupling and finally for asynchronous sequential explicit coupling.

7. After finishing the coordination of the initial data exchanges for explicit thermal-hydraulic coupling, listen to receive the initialization and run status message from each of the simulation codes.

8. After all initialization and run status messages have been received, print a summary of the initialization and run status for each simulation code to the output file and the terminal screen to inform the user of the status of the coupled computation.

9. Check the initialization status and terminate the coupled computation if any of the simulation codes returns an initialization error condition.

10. Check the run status of all of the simulation codes, determine the global run status, and broadcast the global run status flag to all of the simulation codes. Terminate the coupled computation if any of the codes indicate that it does not want to proceed to the transient simulation (i.e., stop after input processing and initialization).

11. If all of the simulation codes indicate that they want to proceed with the coupled computation, call subroutine TRAN.

12. After subroutine TRAN returns, check for an error condition, print a summary of the error condition if needed, and print the end time of the simulation.

13. Terminate the coupled simulation for an error condition. The simulation codes are expected to exit automatically when the end time of the simulation is reached.

14. Shutdown the virtual machine after all simulation codes have exited.

15. End

The preceding list gives a broad outline of the computational flow in procedure PVMEXEC. Each task on the list may encompass many lines in the source code including calls to subroutines and functions

in the PVM software library or other subroutines in the PVMEXEC program source. The data exchanges between the PVMEXEC program and the simulation codes and between the simulation codes is performed by the message passing infrastructure provided by the PVM software library. The PVMEXEC program and each instance of a simulation code are given a PVM task identification number when they are added to the virtual machine. The task identifier is used to direct a message to a specific task. Each message has a message tag that distinguishes it from other messages. The PVMEXEC program and the coupling system it implements use a handshaking protocol where each message sent to a task must be acknowledged by a message from the receiving task with the same message tag. An acknowledgement usually consists of a single data item, the task identifier of the task sending the acknowledgement. This helps keep the simulation codes synchronized. The message tags for messages between the PVMEXEC program and the simulation codes are listed in Appendix A along with a description of the contents of the messages. The message tags for messages between the simulation tasks are assigned by subroutine INPUTD using the information contained in the input file for the PVMEXEC program. The message tags for these messages are sent to the several simulation codes in the messages from the PVMEXEC program that specify the data items that are to be exchanged between the codes.

Because the PVM virtual machine may consist of hosts of different architectures (i.e., a PC, a DEC Alpha, a SUN, etc.) and because integers and real numbers are represented by different bit patterns in the memories of the several different types of architectures, the PVM software defines several different data types. The PVM data types used by the PVMEXEC program are INTEGER4 for four byte integers, REAL8 for eight byte real numbers and STRING for strings of characters. These data types are defined in a PVM header file that is ‘included’ in each of the subroutines in the PVMEXEC program. The PVM software establishes a mechanism to convert the data type into a machine independent representation, XDR format, for the transfer of data between hosts of different architectures. However, this results in a loss of precision for real numbers. The PVM software can also transmit data items without any conversion (RAW format), but this is only meaningful if the two codes exchanging the data are executing on the same architecture. Therefore the PVMEXEC program, and the simulation codes, are expected to determine if they are exchanging data with codes executing on the same or different architectures and send their data in the appropriate format. This determination is performed by calls to subroutines in the PVM software library.

After the PVMEXEC program sends a message to one of the simulation tasks, or broadcasts a message to all of the simulation tasks, it listens to receive an acknowledgement from the receiving task(s). If the receiving task has failed for any reason, the PVMEXEC program would wait indefinitely unless instructed otherwise. The PVMEXEC program was programmed to only wait for a specified amount of time to receive an acknowledgement. A default wait time of 60.0 sec. has been programmed into the PVMEXEC program but the default wait time may be overridden by optional input in the input file. While listening to receive an acknowledgement to a specific message, the PVMEXEC program also listens to receive two other messages. These are messages with message tag 10001 and with message tag 10002. The message with message tag 10001 is sent by the virtual machine and contains the task identifier of simulation code that has failed. The message with message tag 10002 contains the task identifier of a task that has exceeded its wait time while waiting to receive an acknowledgement to a message that it sent to

another task or exceeded its wait time while waiting to receive a message from another task. If either of these messages are received by the PVMEXEC program, the global termination message with message tag 10003 is broadcast to all of the simulation codes and the virtual machine is shutdown after all of the remaining simulation codes have exited. The simulation code sending the message with message tag 10002 is expected to exit without waiting to receive the message with message tag 10003 from the PVMEXEC program and the other simulation codes are expected to check to see if a message with message tag 10003 has been received from the PVMEXEC program while waiting for messages with any other message tag. The simulation codes are also expected to check for a message with message tag 10001 while listening to receive other messages. They are expected to request this message from the virtual machine to designate that the PVMEXEC program has failed and cannot coordinate the virtual machine. A simulation code is expected to exit after receiving a 10003 message or a 10001 message. Subroutine PVMFXREC is the subroutine in the PVMEXEC program that listens to receive message acknowledgements and the task failure messages (messages with message tags 10001 and 10002).

### 2.1.1 Coordination of Initial Data Exchanges

Task 6 in the list of subtasks performed by the PVMEXEC program is to coordinate the initial data exchanges between the codes that are participating in explicit thermal-hydraulic coupling. Because the PVM data exchange methodology is like an telephone party line, only one task should be talking at a time and the other tasks should be listening. The PVMEXEC program coordinates the data exchanges by designating each task in turn as the ‘sending’ task. If a task is not the designated ‘sending’ data, it is a ‘receiving’ task and is expected to listen for messages from the designated ‘sending’ task (if it expects to receive data from the designated ‘sending’ task). Once the ‘sending’ task has sent all of its messages to the other tasks and has received acknowledgements to the messages that it has sent, it must send an acknowledgement to the PVMEXEC program in a message containing its task identifier. This message signifies to the PVMEXEC program that the designated sending task had completed its data exchanges. Likewise, the other tasks are expected to send an acknowledgement to the PVMEXEC program after they have received and acknowledged the messages from the designated ‘sending’ task. The messages from the receiving tasks to the PVMEXEC program designate that the ‘receiving’ tasks has received all of the data that it expects to receive from the ‘sending’ task. Once all of the acknowledgements have been received by the PVMEXEC program, it designates the next task participating in explicit thermal-hydraulic coupling as the ‘sending’ task and the process repeats. The process of designating tasks as the sending task and the others as the ‘receiving’ tasks is repeated until all of the tasks participating in one the types of explicit thermal-hydraulic coupling have had their turn to be the ‘sender’. When all processes participating in one of the types of explicit thermal-hydraulic coupling have been designated as the ‘sending’ task, the PVMEXEC program sends a final message containing a task identifier of zero. This designates to all of the processes participating in explicit thermal-hydraulic coupling that all tasks have had their chance to send their data and that the tasks should proceed with their initialization. This process is repeated for each type of explicit thermal-hydraulic coupling. The lists of tasks participating in each type of explicit thermal-hydraulic coupling are determined in subroutine INPUTD.

The initial data exchanges for explicit sequential thermal-hydraulic coupling use the procedure described above except that the data exchanges are performed twice. The first time, only the tasks sending ‘volume’ data should send their messages (and conversely, only those tasks receiving ‘volume’ data from the designated ‘sending’ task should listen to receive messages from the sending task), and the second time only junction data should be exchanged. This order of sending ‘volume’ data first and then ‘junction’ data is used so that the ‘volume’ data that is exchanged may be used to determine the appropriate ‘junction’ data that is to be exchanged in the second exchange (i.e., most thermal-hydraulic simulation codes use the ‘donor’ method of determining junction properties from ‘volume’ properties). The coupled tasks are expected to determine whether they are sending ‘volume’ or ‘junction’ data for explicit sequential thermal-hydraulic coupling from the information contained in the data specification messages they receive from the PVMEXEC program.

## 2.2 Subroutine INPUTD

Subroutine INPUTD reads the input file for the PVMEXEC program and builds a database from the information in the input file. Using the information in the database, it configures the virtual machine, it executes the several instances of the simulation codes on the hosts of the virtual machine, and it tells each simulation code what data they are to exchange with the other simulation codes. The order of computation is as follows:

1. Read input file
2. Process input data including assignment of message tags
3. Configure and start virtual machine
4. Execute simulation tasks on virtual machine
5. Specify transient simulation
6. Inform the simulation tasks about the data to be exchanged with other tasks

These tasks will be described in the following sections. The input file of the PVMEXEC program is divided into sections by reserved first level keywords. These first level keywords are independent of the particular simulation being performed. The data in the input file define another set of second level keywords that depend on the particular simulation being performed. The first level keywords divide the input file into five sections. The sections may occur in the input file in any order but they are processed in a particular order independent of their order in the input file. Before processing in input data, subroutine INPUTD first counts the number of lines in the input file, allocates a buffer large enough to hold the input lines, reads the input file into the buffer, line by line, and prints the input buffer to the output file of the PVMEXEC program. All subsequent processing is performed in the copy of the input file held in the input buffer. Comments begin with the reserved character “#” and must be the first non-blank character in a line.

### 2.2.1 Processing of Input Data

#### *2.2.1.1 Description of Virtual Machine*

The first section of the input file to be processed is an optional section of the input file and describes the virtual machine to be used for the coupled computation. This section of the input file begins with the reserved keyword ‘virtual’ and is terminated by another of the first level keywords or the end of the input file. Subroutine INPUTD searches for the keyword ‘virtual’ and then for the end of this section in the input file contained in the input buffer. If this section is found in the input file, it processes this section of the input file line by line by reading the first work on each line and comparing it to the reserved keywords ‘wait’ and ‘writes’. The input lines are read by calls to subroutine GETWORD that parses the line and returns the blank delimited words. The input for subroutine GETWORD is the particular line to be read and the position of the word in the line (the first word on the line is word zero [like the C language]). Subroutine GETWORD returns the word in the specified position, and the location in the line for the start of the next word. The line beginning with the keyword ‘wait’ contains the global wait time and the debugging flag and the line beginning with the word ‘write’ specifies an alternate output file name for output from the virtual machine. All other lines in the ‘virtual’ section of the input file specify the names of the hosts to be used in the coupled computation. These lines are counted to determine the number of hosts in the virtual machine. Once the number of hosts in the virtual machine have been counted, a number of data arrays whose length are the number of hosts in the virtual machine are allocated. The values in these arrays are assigned from the information in the ‘virtual’ section of the input file. The first word on each line not beginning with the reserved keyword ‘wait’ or ‘write’ is saved in a list of the names of the hosts in the virtual machine, and the remainder of each line is saved as a character string that contains parameters for the particular host in the virtual machine. These parameters are defined in the PVM manual and specify such things as where in the file system the executable files can be found, which directory is to be the working directory for the computation, etc. (see the PVM manual for a complete list of the parameters that may be specified in this string). The parameter data is saved as a string and is passed to the virtual machine without being parsed by the PVMEXEC program. If the ‘virtual’ section is not found in the input file, it is assumed that the host executing the PVMEXEC program is to be the only host used in the coupled simulation, that the executable files are to be found on the directory from which the PVMEXEC program was executed, and that the working directory for the coupled computation is the same directory. The data arrays are assigned by using environmental routines to determine the name of the host on which the PVMEXEC program is executing and the name of the directory on that system from which the PVMEXEC program was executed. Once the data arrays for the virtual machine have been populated, the configuration of the virtual machine is printed to the output file of the PVMEXEC code. These data arrays constitute the ‘machine’ database for the coupled simulation. This completes the processing for the ‘virtual’ section of the input file.

#### *2.2.1.2 Types of Transient Simulation*

The second section of the input file for the PVMEXEC program is an optional section of the input file that describes the type of transient simulation being performed and a unique name for the simulation. The simulation may be an initial run or may be a restart of a previously performed simulation. A unique

name for the simulation may also be specified to distinguish it from other simulations. The name is to be written to the restart files by the simulation codes for an initial run or is to be compared to the name on the restart files for a restart run to ensure that the correct restart file is being used. This section of the input files begins with the keyword ‘simulation’ and ends with another first level keyword or the end of the input file. Subroutine INPUTD first searches the input file to determine if this section has been specified by the user in the input file. If this section is not found in the input file, the start time and the restart time variables are set to a value of zero and the character string for the name of the simulation is set to 80 blanks. If this section is found in the input file, the first word of each line is read to find the second level keywords ‘restart’ and ‘name’. The line beginning with the keyword ‘name’ specifies the unique name of the simulation. The data on this line after the first word are read into the character string specifying the name of the simulation. If the first word on a line is the keyword ‘restart’, the third word is read and the value converted to a real number specifying the restart time for the run. If this keyword is not found, the restart time remains at its initial value of zero. If the restart time is found in the input file, the value of the start time is set to the value of the restart time.

After all of the lines in the simulation section of the input file have been processed to find the keywords ‘name’ and ‘restart’, they are processed again to find the keyword ‘start’. If a line with this keyword is found in the input file, the third word on this line is read and converted into a real number specifying the start time for the simulation. At the end of this second processing step, the start time, restart time, and name variables should all have values, either their defaults values, or the values specified in the input file. Once the processing of the ‘simulation’ section has been completed, the ‘simulation’ data for the run are printed to the output file of the PVMEXEC program. This ends the processing of the ‘simulation’ section of the input file.

#### *2.2.1.3 Processes In Virtual Machine*

The third section of the input file to be processed by subroutine INPUTD is the ‘processes’ section of the input file. This section of the input file is delimited by the first level keyword ‘processes’ and is terminated by a line containing another first level keyword or the end of the input file. This section of the input files is divided into subsections delimited by the names of the hosts found in the ‘virtual’ section of the input file. This section of the input file is processed twice. Each line is read to determine the number of ‘processes’, or tasks in PVM parlance, in the simulation being performed by the virtual machine. Lines beginning with the names of the hosts in the virtual machine are skipped when counting the tasks in the virtual machine as well as lines containing the reserved keywords ‘uses’ and ‘writes’ as the second word in the line in the input file. All other lines are counted as containing the specification for tasks in the virtual machine. Once the number of tasks has been determined, several data arrays are allocated, each array having the number of tasks as its length and whose names begin with the string ‘proc_’. These arrays constitute the ‘processes’ database.

The ‘processes’ section of the input files is then read again to populate the data arrays. The lines specifying the tasks, i.e., lines not containing the names of hosts as their first word or the reserved keywords ‘uses’ or ‘writes’ as their second word, are parsed into four pieces. The first word on the lines specifies a unique name for the PVM task, the second word must be the reserved keyword ‘synchronous’

or the reserved keyword ‘asynchronous’, the third word is the name of the executable file for the simulation code, and the remainder of the line is read into a character string that is used as the command line to the executable when the task is executed as a PVM task. The string ‘-PVM’ is appended to the string containing the command line parameters so that the task can determine if it is being executed as part of a coupled simulation. One of the requirements for the modifications to the simulation codes (to implement the PVM Application Coupling Interface for participation in a coupled simulation) is for the simulation code to determine whether or not it is being executed under the control of the PVMEXEC program. This requirement was specified so that the same executable file may be executed in stand-alone mode or coupled mode, thus simplifying the V&V for simulation codes that have been modified for coupled execution. The names of the tasks become second level keywords in the next section of the input file. Once a line specifying a task has been found and parsed, the ‘processes’ section is scanned to find any lines beginning with the name of the task as the first word on the line and the reserved keywords ‘uses’ or ‘writes’ as the second word on the line. If such lines are found, the number of threads to be used by the task is set from the value on the line containing the keyword ‘uses’ and the name of the PVM output file is set from the character string on the line containing the reserved keyword ‘writes’. Once all of the lines specifying the tasks have been read and parsed, the data arrays containing the name of the host on which to execute a task, the name of the task, the type of task (i.e., synchronous or asynchronous), the name of the executable for the task, the name of the PVM output file for the task, the number of threads to be used for the task, and the command line parameters for each task have been determined. The name of the output PVM output file for a task and the number of threads for a task are optional input and contain a string of blanks and zero, respectively, if not specified in the input file. Once the processing of the input data in the ‘processes’ section of the input file have been completed, the processed data are written to the output file of the PVMEXEC code. This completes the processing of the ‘processes’ section of the input file.

#### *2.2.1.4 Data Exchanges for Coupled Simulation*

The next section of the input file is the ‘messages’ section of the input file. This section is divided into subsections for the several types of coupling. The subsections are delimited by the reserved keyword ‘explicit’ for explicit thermal-hydraulic coupling, the reserved keyword ‘semi-implicit’ for semi-implicit thermal-hydraulic coupling, the reserved keyword ‘kinetics’ for kinetics coupling’, and the reserved keyword ‘control’ for control systems coupling. The lines in this section of the input file are read and parsed to determine the number of messages that are to be exchanged between the several tasks executing in the virtual machine. Lines that contain the keyword ‘sends’ or ‘receives’ as the second word on the line are counted and contain the specification for the data to be exchanged between tasks. Once the number of messages has been determined, a set of data arrays is allocated, each array of the length corresponding to the number of messages, and whose names begin with the string ‘msg_’. These data arrays constitute the ‘massages’ database.

Once the arrays for the messages have been allocated, the data arrays are populated by reading the message specifications. Each line in the ‘messages’ section is read one at a time. If the first work on the current line is one of the keywords that delimit the several subsections, e.g., ‘semi-implicit’, ‘explicit’, etc., a flag that specifies which subsection is currently being processed is reset based in the keyword found, the

starting location in the input buffer for the current subsection is initialized, and the ending location in the input buffer for the previous subsection is set. If the first work on the current line is not a reserved keyword, the second word must be the keyword ‘sends’, the keyword ‘receives’, the keyword ‘awaits’, or the keyword ’precedes’. If the second word is the keyword ‘sends’, the send flag is set to zero; if the second word is the keyword ‘receives’, the send flag is set to one. If the second word is not ‘sends’ or receives’, subroutine INPUTD proceeds to the next line in the ‘messages’ section of the input file. If the keyword is ‘sends’ or ‘receives’, the third word is read and the values in several of the data arrays are set. The first word on the line is loaded into the array msg_proc. The value of the send flag is loaded into the array msg_type. The third word is loaded into the array msg_fromto, and the remaining words on the line are loaded into the array msg_data. This process divides the lines that specify the data that is to be exchanged between any two processes into four pieces. Subroutine INPUTD then determines whether the message is between synchronously controlled or asynchronously controlled processes. The process database is scanned for the type of process, synchronous or asynchronous, of the tasks named in the msg_proc and msg_fromto arrays for the current message. If both tasks are asynchronous tasks, the message type is asynchronous, otherwise the message type is synchronous. The message type is loaded into the variable msg_stype where a value of zero denotes a synchronous message and a value of one denotes an asynchronous message.

Once all of the lines in the ‘messages’ section of the input file have been read, they are scanned two more times to determine the wait time for a message and to set the flag for sequential explicit thermal-hydraulic coupling. The message database is processed one message at a time. The wait time for each message is first set to the global wait time, either the default wait time or the time specified on the ‘wait’ card in the ‘virtual’ section of the input file. Next the message type, i.e., ‘explicit’, ‘semi-implicit’, etc., is determined from the msg_stype variable for the message. Using the message type, the particular subsection of the ‘messages’ section is read one line at a time to find a line that matches the template consisting of the value of msg_proc for the first word, the reserved keyword ‘awaits’ for the second word, and the value of msg_fromto for the third word. If such a line is found, then the wait time for the message, variable msg_wait, is set to the value of the fourth word on that line. This process is repeated for each message in the message database.

The ‘explicit’ subsection of the ‘messages’ section of the input file is scanned to find lines whose second word is the keyword ‘precedes’. If such a line is found, the process database is scanned to find the process name that matches either the first word or the third word on the line containing the keyword ‘proceeds’. The proc_extype variable for this process is set to one denoting that the process is participating in sequential explicit coupling. The default value of proc_extype is zero denoting that the process is not participating the sequential explicit thermal-hydraulic coupling. The message database is also scanned to find the messages whose values of msg_proc and msg_fromto match the first and third words on the line containing the keyword ‘proceeds’. When such a message is found in the message database, the variable msg_extype is set to zero for the message denoting that the value in the variable msg_proc was named first on the line containing the keyword. If the values of msg_proc and msg_fromto match the third word and first word on the line containing the keyword ‘precedes’, then the value of msg_extype is set to one for the message denoting that the value in the variable msg_proc was named second in the line containing the

keyword. The default value for the variable msg_extype is -1 denoting that the message is for parallel explicit thermal-hydraulic coupling. This completes the processing of the data in the ‘messages’ section of the input file.

#### *2.2.1.5 Time Step and Output Control for Coupled Simulation*

The last subsection of the input file contains the data to control the time step size for synchronous coupling, the explicit coupling interval for asynchronous processes, the times at which output data are produced and the end time for the simulation. This section of the input file begins with the keyword ‘timesteps’. Subroutine INPUTD counts the number of time step lines and allocates data arrays of the required length. The lines in the ‘timesteps’ section are read one at a time. Each line should contain eight data items and are loaded into the proper array elements one data item at a time for each line. Once all of the lines in the ‘timesteps’ section of the input file are processed, the data arrays are printed to the output file of the PVMEXEC program. This ends the processing of the ‘timesteps’ section of the input file. All further processing is done using the data in the ‘machines’ database, the ‘processes’ database, and the ‘messages’ database.

#### *2.2.1.6 Assignment of Message Tags*

The PVM message passing methodology uses an identifier for each message sent from one task to another task to distinguish it from other messages between the same two tasks. The message identifier is called a message tag. The message tags for messages between the PVMEXEC program and the tasks executing on the virtual machine have been determined by the programmer of the PVMEXEC program. The message tags for messages between simulation tasks could have been assigned by the user in the input file but are determined by the PVMEXEC code because they are of no significance and can be assigned arbitrarily. The message tags are assigned after the ‘timesteps’ section of the input file has been processed. The message database is scanned one message at a time to find the ‘send’ messages. If a message is a ‘send’ message, i.e., its value of msg_type is zero, its message tag, variable msg_tag, is set to the next available value. The values of the message tags begin with a value of one and are incremented by one. Once the message tag has been set for a ‘send’ message, the message tag for its corresponding ‘receive’ message must be set to the same value. The message database is searched for a message of the same message type, i.e., type ‘explicit’, semi-implicit’, etc., whose msg_proc is the same as the value of msg_fromto of the ‘send’ message, and whose value of msg_fromto is the same as the value of msg_proc for the ‘send’ message. Once such a match is found, its message tag is set to the same value as the message tag for the ‘send’ message. Once the message tag for each message in the ‘messages’ database have been determined, the data arrays that constitute the ‘message’ database are printed to the output file of the PVMEXEC program.

### 2.2.2 Assembly of the Virtual Machine

Once all of the input data has been processed, the virtual machine can be assembled from the hosts specified in the input file. This is accomplished by a call to the PVM library subroutine ‘pvmfstartpvmd’. This subroutine requires the name of a ‘hostfile’ as one its parameters. This ‘hostfile’ is assembled using

the data in the ‘machines’ database. The name of the hostfile is assembled from a fixed string and the process ID of the PVMEXEC program found using the system environmental routine GETPID. The ‘pvmfstartpvmd’ returns a status flag that indicates whether the PVM daemon process was initiated on the hosts named in the ‘hostfile’. If an error condition is detected, the coupled simulation is terminated. Next the PVMEXEC program is enrolled in the virtual machine by a call to the PVM routine ‘pvmfmytid’. This returns the PVM task identifier of the PVMEXEC program.

### 2.2.3 Execution of Simulation Tasks

Once the virtual machine has been assembled and the PVMEXEC program enrolled in the virtual machine, the several simulation tasks are executed on the virtual machine. The method for the initiation of the simulation task depends upon whether the user has decided to redirect the standard output file ‘stdout’ (the standard output is normally written to the terminal screen) for the task to a separate file or whether the user allows the standard output file for the simulation task to be written to the standard output file for the virtual machine. By default, the standard output file for the virtual machine is written to the directory specified by the ‘TMP’ environmental variable with the file name of pvml.’uid’, where the ‘uid’ is the user identification number of the user of the virtual machine. The output for the virtual machine may be redirected to the file specified in the line in the ‘virtual’ section of the PVMEXEC input file that begins with the keyword ‘write’. The user may optionally specify the file to which the standard output from a particular simulation task is written by including a line in the ‘processes’ section of the PVMEXEC input file that contains the keyword ‘writes’ as the second word on the line with the name of the process writing the file as the first word on the line and the name of the file as the third word. If the standard output from a simulation task has not been redirected to its own output file, the task is initiated by a call to subroutine PVMSPAWNX. This routine starts the particular task on the appropriate host and returns a task identifier for the task and a status flag. If no errors are indicated in the status flag, a message with message tag 7001 is sent to the simulation task. This message contains a single data item, the task identifier of the PVMEXEC program. After sending the message with message tag 7001, the PVMEXEC program listens to receive an acknowledgement for the task receiving the message. Once it receives the acknowledgement, it proceeds to start up the next simulation task.

If the user has elected to redirect the standard output from a simulation task to its own output file, the initiation of the simulation task is a more complicated, two step process. Because the standard output from a simulation task can only be redirected by its parent process, each simulation task whose output is to redirected must have a different parent task. This is accomplished by interposing another PVM task between the PVMEXEC program and the simulation task. The PVMEXEC program initiates this task, contained in the executable file for the PVMCATCHOUT program, and then directs the PVMCATCHOUT task to initiate the simulation task while redirecting its standard output to a separate file. The PVMEXEC program initiates the PVMCATCHOUT task using a call to the PVM library subroutine ‘pvmfspawn’. Once the PVMCATCHOUT task has been initiated, the PVMEXEC program sends it a message with message tag 7000 containing the data needed to initiate the simulation task. When the PVMCATCHOUT task receives the message with message tag 7000, it initiates the simulation task specified in the message using subroutine PVMSPAWNX and sends it a message with message tag 7001

containing the task identifier of the PVMEXEC task (the task identifier of the parent of the PVMCATCHOUT task). The task identifier of the parent task is found by a call to PVM library routine ‘pvmfparent’. Once the simulation task receives the task identifier of the PVMEXEC task, it acknowledges the receipt of the 7001 message to the PVMCATCHOUT task, and the PVMCATCHOUT task acknowledges the receipt of the 7000 message to the PVMEXEC task with a message containing the task identifier of the simulation task. Notice that the simulation task receives the task identifier of the PVMEXEC task in a 7001 message regardless of whether its parent is the PVMEXEC task or the PVMCATCHOUT task.

Once all of the simulation tasks have been executed on the virtual machine, the PVMEXEC program performs several housekeeping tasks. First it compares the architecture of the host on which the PVMEXEC program is executing to the architecture of the host on which the several simulation tasks are executing. The architectures of the several hosts are found by calls to PVM library routine ‘pvmfconfig’ and ‘pvmftidtohost’. The comparison is used to set the variable proc_dtype for each simulation task in the coupled simulation. This variable determines if ‘raw’ mode or ‘xdr’ mode are used by the PVMEXEC program when sending messages to the several simulation tasks.

Next, the PVMEXEC program counts the number of tasks that are participating in each of the four types of explicit thermal-hydraulic coupling, allocates data array using these sizes, and builds lists of the task identifiers of the simulation tasks participating in each type of explicit thermal-hydraulic coupling. It also builds two lists of the data transmission mode (i.e., ‘raw’ or ‘xdr’) for the tasks participating in explicit asynchronous thermal-hydraulic coupling. It builds a master list of all of the processes that are participating in either type of explicit asynchronous thermal-hydraulic coupling and a master list of all tasks that are participating in synchronous coupling of any type (i.e., explicit thermal-hydraulic coupling, semi-implicit thermal-hydraulic coupling, etc.). Finally it builds a list of the data transfer mode to the list of tasks that are synchronously coupled. These lists are built from the data in the ‘processes’ database and are built once and used during the transient advancements.

### 2.2.4 Task Status

The PVMEXEC program needs to know the execution status of the several simulation tasks after they have been initiated so that the coupled computation may be terminated gracefully if any of the simulation tasks fail or encounter an unrecoverable error during their execution. The status of tasks in the virtual machine may be determined by a call to PVM library routine ‘pvmnotify’. This routine will send a message with the specified message tag in response to the specified event for any of the task specified. The PVMEXEC program requests that the virtual machine send a message with message tag 10001 if any of the simulation tasks exit abnormally. This allows the PVMEXEC program to shutdown the coupled simulation gracefully if any of the simulation tasks fail catastrophically.

### 2.2.5 Specification of Transient Simulation

Tbe PVMEXEC program begins a coupled simulation by telling the simulation tasks what type of transient simulation is to be performed. It broadcasts a message with message tag 1 to all of the simulation

tasks. This message contains the start time, the restart time, and the name of the simulation. After sending this message, the PVMEXEC program listens to receive acknowledgements from all of the simulation tasks. The acknowledgement message contains the restart status of the simulation task along with other information that is used to write error messages if the simulation task cannot perform the restart at the specified time. The global restart status is determined from the restart statuses of the simulation tasks and is sent to the simulation tasks in another message with message tag 1. The PVMEXEC program waits to receive an acknowledgement from the simulation tasks before sending the global restart status message to the next simulation task.

After sending the global restart status message to each of the simulation tasks, the PVMEXEC program sends a message with message 2 to all of the simulation tasks. This message specifies the number of threads that the task is to use during its execution. The default value of zero indicates that the task is to determine the number of threads and a non-zero value indicates that the user has specified the number of threads to be used by the task in the input file for the PVMEXEC program.

The last bit of housekeeping is for the PVMEXEC program to send the simulation tasks the wait time and debugging flag in a message with message tag 1000. The wait time is the wait time to be used for messages between the simulation tasks and the PVMEXEC program (the wait times for data exchanges between the simulation tasks is specified for each message from the ‘message’ database) and the debug flag may be used to activated extra printout for debugging or any other purpose.

### 2.2.6 Specification of Data Exchanges

Finally, the PVMEXEC program tells each of the simulation tasks what data it is to exchange with the other simulation tasks. The specification of data exchanges is contained in triples of messages for each type of coupling, explicit thermal-hydraulic coupling, semi-implicit thermal-hydraulic coupling, kinetics coupling, and control systems coupling. Each triplet of messages begins with the number of send and receive messages for that type of coupling followed by messages the specify the contents of the send and receive messages.

The first triplet of messages begins with a message with message tag 1001. This message contains the number of send and receive messages that are to be used by the first task in the list of tasks participating in explicit thermal-hydraulic coupling (this list was build in Section 2.2.3). The message with message tag 1001 is followed by a series of messages with message tag 1002. These messages contain the following data; the message tag to be used in sending the data, the task identifier of the task to which the message is to be sent, the transient type of the message (i.e., synchronous or asynchronous), the subtype of explicit thermal-hydraulic coupling that the message facilitates (i.e., parallel, leader, or follower), the wait time for the acknowledgement to the message, the number of characters in the following string, followed by a string variable of the specified length that designates the data items to be sent. The string variable is the fourth piece of the message specification lines in the ‘messages’ section of the input file (see Section 2.2.1.4). It is the responsibility of the simulation task to understand the contents of the string variable as the names and locations of variables in its simulation model.

The PVMEXEC program sends the appropriate number of messages with message tag 1002 and then begins to send messages with message tag 1003. These messages specify the data that are to be received from other simulation tasks for explicit thermal-hydraulic coupling. The messages contain the following data; the message tag of the message that is to be received, the task identifier of the task from which the message is to be received, the transient type of the message, the subtype of explicit coupling that the message facilitates, the wait time for receiving the message, the length of the following string, followed by a string variable of the specified length. The string variable designates the variables that are to be received in the message. The PVMEXEC program sends the requisite number of receive message specification messages.

The next triplet of messages begins with a message with message tag 1004. This message specifies the number of send and receive messages that the tasks will send or receive for semi-implicit thermal-hydraulic coupling. The specifications for send messages are sent in messages with message tag 1005 and the specifications for receive messages are sent in messages with message tag 1006. The send specification messages contain the following data; the message tag for the message, the task identifier of the task to which to send the message, the wait time for the acknowledgement to the message, the length of the following string variable, followed by the string variable that designates the variables to be sent. The format of the receive message specification is as follows: the message tag of the message that is to be received, the task identifier of the task from which to receive the message, the amount of time to wait to receive the message, the length of the following string, followed by the string variable designating the variables to be received.

The triplet of messages for the specification of the data that is to be exchanged for semi-implicit thermal-hydraulic coupling is followed by the triplet of messages for the specification of the data that is to be exchanged for kinetics coupling. These messages use message tags 1007, 1008, and 1009. The format of the messages is the same as the format for the specification of the data be exchanged by semi-implicit thermal-hydraulic coupling.

The last triplet of messages are for the specification of data to be exchanged for control systems coupling. These messages use message tags 1010, 1011, and 1012, respectively. The format for the messages is the same as the format for the specification of semi-implicit coupling messages.

This completes the description of subroutine INPUTD.

## 2.3 Subroutine TRAN

Subroutine TRAN controls the transient simulation being performed by the coupled simulation codes executing on the virtual machine. It is called by the main program after it has been determined that all of the simulation tasks can proceed to the transient phase of the coupled simulation and the global run flag has been transmitted to and has been acknowledged by the several simulation tasks.

The order of computation is as follows:

1. Write header for status information

2. Call subroutine DTSTEP

3. Listen to receive status message from synchronously coupled tasks

4. Determine global success status (i.e., successful time step advancement)

5. Broadcast global success status to all tasks participating in synchronous coupling

6. Write status information to PVMEXEC output file and terminal screen if indicated

7. Return to step 2 if transient simulation not finished

8. Write final status information to PVMEXEC output file and terminal screen

9. Return to subroutine PVMEXEC

Subroutine TRAN begins by writing a header for the execution statistics to the output file of the PVMEXEC program and to the terminal screen followed by the initial status of the coupled computation. The status information includes the wall clock time since the coupled simulation began, the initial simulation time, the initial time step size for synchronously coupled tasks, and the number of advancements since the simulation was begun. The initial time step size and initial number of advancements are written as zero on the initial status line. The wall clock time on the initial status line shows the time spent in input processing and initialization.

Subroutine TRAN then calls subroutine DTSTEP. Subroutine DTSTEP will be described in the following section. Once subroutine DTSTEP returns, subroutine TRAN listens to receive a message with message tag 10000 for all of the tasks participating in synchronous coupling. The message contains the status flag for synchronous coupling. Subroutine TRAN determines whether the advancement was successful or not (i.e., the global success status). If the advancement was not successful (i.e., one or more of the simulation tasks want to repeat the time step), subroutine TRAN determines the type of repeat (i.e., repeat with the same time step size, repeat with a smaller time step size, or stop) and the task that is requesting the repeat. After determining the global success status, it broadcasts the global success status to all of the simulation tasks participating in synchronous coupling.

After receiving the acknowledgements to the global success status, subroutine TRAN determines if enough wall clock time has passed to write the next transient status line to the output file of the PVMEXEC program and the terminal screen. It writes the transient status line if needed (preceded by another header if enough status lines have been written to the screen) and then calls subroutine DTSTEP to determine the time step size for the next attempted time advancement, either the time step size for the next time step advancement or the time step size for a repeated attempt for a failed advancement. Some types of time step failure, such as a velocity flip-flop, use the same time step size on the repeated advancement. In this case, subroutine DTSTEP returns the same time step size as for the first attempt of the advancement.

In any case, subroutine DTSTEP determines the time step size for the next attempted advancement and instructs the simulation code to begin the next attempted advancement.

The process of calling subroutine DTSTEP to determine the time step size and listening to receive the transient status in messages with message tag 10000 is repeated until either an unrecoverable error occurs or until the end time of the coupled simulation is reached. At the end of the transient, however it ends, subroutine TRAN writes a final status line to the output file of the PVMEXEC program and the terminal screen.

## 2.4 Subroutine DTSTEP

Subroutine DTSTEP determines the time step size that is to be used for synchronously coupled tasks, determines the explicit coupling interval for asynchronously coupled tasks, and determines the times at which output is to be produced by the coupled tasks, all subject to the time step controls contained in the 'timesteps' section of the input file for the PVMEXEC program. The data on the time step cards divide the simulation into a number of intervals, with the data on each time step card defining how the time step size and output times are determined for each interval. Within an interval, the code chooses the global time step by halving and doubling the current time step size to find the time step that is less than the smallest time step requested by the simulation codes. The halving and doubling are limited by the maximum and minimum time step sizes specified by the user on the time step cards. The initial guess of the time step size is the maximum time step size of the first time step interval. In addition to determining the time step size, subroutine DTSTEP determines the times within each time step interval at which to produce output, both printed and plottable output, for synchronously coupled tasks, the times for data exchanges for asynchronously coupled tasks, and the times for the production of restart information for all of the simulation tasks. The time step size is chosen so that the simulation time at the end of a series of advancements is equal to the minimum of the current set of output times for the current time step interval, the time at the end of the explicit coupling interval, and the time at the end of the current time step interval. Because the time steps must be chosen in such a way that the simulation time is equal to the time described in the previous sentence, the time step size at the end of a time step interval may not be an integer fraction of the maximum time step size, i.e., may not be chosen by halving and doubling.

The logic in subroutine DTSTEP is very complicated because the routine was adapted from the routine of the same name in the RELAP5-3D program. The routine uses a large number of GOTO statements that makes it difficult to explain the logic in a straightforward manner. The main subsections of subroutine DTSTEP are explained in the following paragraphs but the sections are not necessarily in the order that they are encountered in the subroutine.

Subroutine DTSTEP begins by setting some local variables during the first call to the routine. Remember that the routine is called at the beginning of the first transient advancement and subsequently at the end of every advancement. The call at the beginning of the first advancement is to emulate the call at the end of the nonexistent previous successful advancement. It sets the output times for the current time step interval to the beginning time of the transient and jumps to the section of the subroutine that

coordinates explicit data exchanges bypassing the backup time step selection logic for a repeated time step advancement.

### 2.4.1 Coordination of Data Exchanges for Explicit Coupling

First, subroutine DTSTEP coordinates the exchange of data between codes that are coupled explicitly. There are four sets of control messages corresponding to each type of explicit coupling. Tasks that are participating in synchronous parallel explicit coupling are sent a series of messages with message tag 8002. Each message names one task in turn so that each task can send its data when it is named in the message with message tag 8002. These data exchanges are similar to the data exchanges performed during the initialization phase of the coupled computation as explained in Section 2.1.1. Once the data exchanges for synchronous parallel explicit coupling have been completed, the data exchanges for asynchronous parallel explicit coupling are coordinated by a series of messages with message tag 8003. Then the data exchanges for synchronous sequential explicit coupling are coordinated by messages with message tag 8004, and finally, the data exchanges for asynchronous sequential explicit coupling are coordinated by a series of messages with message tag 8005.

### 2.4.2 Time Step Selection and Output Control

The time step and output control logic described in the following paragraphs is executed every time subroutine DTSTEP is called.

After the explicit data exchanges have been coordinated, subroutine DTSTEP checks to determine if the transient simulation is finished (finished because the end time has been reached). If the transient simulation is not finished, the output times, coupling interval time and restart information times are updated if the current simulation time is equal to any of the output times, etc. A list is then built of the simulation tasks needing a new set of output times. The new set of output times, etc. are broadcast to the simulation tasks in the list in a message with message tag 8000. After receiving the acknowledgements to this message, subroutine DTSTEP chooses a preliminary value of the next global time step size based on the halving or doubling subject to the constraint that the end time of the time step interval cannot be exceeded and that the last two time steps in an interval should be of the same size instead of having a large time step followed by a small time where the small time step is needed so that the simulation time exactly equals the time at the end of the time step interval. Once the preliminary global time step size has been chosen, subroutine DTSTEP listens to receive a message with message tag 8001 from each of the tasks participating in synchronous coupling. After receiving the desired time step size from all of the synchronously coupled tasks, a new global time step size is computed as the largest time step that can be computed by repeatedly halving the preliminary global time step size such that it is smaller than the minimum of the desired time steps. This value of time step size along with the current values of the output times are sent to the tasks participating in synchronous coupling in a message with message tag 8001. The current output times are included in this message so that output can be requested at the end of each time step. This option can be activated by a control word on the time step cards. After receiving the acknowledgments for the tasks receiving the 8001 message, the current time and current advancement count are incremented and the subroutine returns to subroutine that called it (subroutine TRAN).

### 2.4.3 Backup Section of Subroutine DTSTEP

The control logic for failed time step advancements is executed whenever subroutine DTSTEP is at the end of an attempted advancement, i.e., not for the first call to the routine but for every subsequent call to the routine. This logic is executed before the common section of the routine as described in the previous subsection of this report.

The logic begins by checking the fail flag determined in subroutine TRAN. The fail flag is set to true if the value of the global success flag is greater than zero. If the fail flag is true, the simulation time is decremented by the value of the current time step, and PVMEXEX listens to receive messages with message tag 8001 from the coupled tasks. These messages contain the time step size that the coupled processes want to use for the repeated advancement. The values in the 8001 messages are processed in the same manner for a failed attempted advancement as they are processed for a successful advancement. (see Section 2.4.2). The simulation will be terminated if the potential time step size is less than the global minimum time step size and the global success flag is greater than one. A value of the global success flag of one indicates that an error was encountered but that the time step advancements can continue at the global minimum time step size. The control logic then jumps to the common section of subroutine DTSTEP where the time step size is broadcast to the simulation tasks participating in synchronous coupling.

### 2.4.4 Preliminary Time Step Selection for Successful Advancement

If the fail flag is false, the current time step size is doubled unless this is the first time step after the time step has been halved. If this is the first time step after a time step that has been halved, the time step size remains constant and the count of the number of halving and doubling is decremented by one. The time step can only be doubled when the count is even. When the time step is doubled, the count is halved. The count is the number of time steps needed to fill an interval equal to the maximum time step size. This is done because the output times are specified as the number of intervals between output times measured in the maximum time step size (i.e., the output frequency).

Subroutine DTSTEP then checks to determine if the simulation is entering a new time step interval. If the current time is the end time for the current interval, the time step interval counter is incremented by one unless the current time step interval is the last time step interval. If the end of the last time step interval has been reached, subroutine DTSTEP sets the end flag and returns to subroutine TRAN.

# 3 All PVMEXEC Programming Units

This section describes all of the programming units that comprise the PVMEXEC program. They are described in alphabetical order. The F90 modules and comdecks that are ‘used’ or included’ in the programming units are also listed and described.

## 3.1 Comdeck BYTE4

Comdeck BYTE4 defines a global constant using a parameter statement.

## 3.2 Module CONTRL

Module CONTRL declares the variables that are used in the control of the transient computations and the determination of the global time step size.

## 3.3 Subroutine DTSTEP

Subroutine DTSTEP has been described in detail in Section 2.4. It controls the production of output and the computation of the global time step size. It also coordinates the data exchanges for explicit thermal-hydraulic coupling.

## 3.4 Subroutine GETSEC

Subroutine GETSEC determines the time in seconds since the beginning of the common era (CE). It calls the F90 date and time routine and converts the values returned by that routine into seconds. The routine is used to determine when to write the status line to the output file of the PVMEXEC program and to the terminal screen.

## 3.5 Subroutine GETWORD

Subroutine GETWORD parses a character string into blank delimited words and returns the word specified in the input parameters. It also determines the character position in the string of the beginning of the next word in the string. Subroutine GETWORD is used to read the lines in the input file of the PVMEXEC program.

## 3.6 Subroutine GNINIT1

Subroutine GNINIT1 is based on the routine of the same name in the RELAP5-3D program. It reads the command line and determines the names of the input and output files for the PVMEXEC program. It opens the input file for reading and the output file for writing.

## 3.7 Subroutine IGNSG15

Subroutine IGNSG15 is a C subroutine that tells the PVMEXEC program to ignore signal 15 from the operating system. Signal 15 must be ignored while the standard output from the simulation tasks is redirected to separate files.

## 3.8 Subroutine INPUTD

Subroutine INPUTD has been described in detail in Section 2.2. It reads the input file, configures and builds the virtual machine, executes the several instances of the simulation codes on the nodes of the virtual machine, and tells each executing task what data it is to exchange with the other tasks executing in the virtual machine.

## 3.9 Module MACHINE

Module MACHINE contains the variables and data arrays that contain the information describing the virtual machine. These variables and data arrays constitute the ‘machines’ database described in Section 2.2.1.1.

## 3.10 Module MESSAGE

Module MESSAGE declares the variables and data arrays that contain the information that describes the data that is to be exchanged between the simulation tasks. These variables and data arrays constitute the ‘messages’ database described in Section 2.2.1.4.

## 3.11 Subroutine NUMWORDS

Subroutine NUMWORDS counts the number of blank delimits words in a character string. It is used to determine if a particular input line has the correct amount of data.

## 3.12 Subroutine POLL

Subroutine POLL is used to receive the acknowledgements to a message that has been broadcast to all of the simulation tasks. It receives the acknowledgements and reports to the calling routine whether or not all of the tasks have responded in the specified wait time. Subroutine POLL is used because the replies may be returned by the simulation tasks in a random order.

## 3.13 Module PROCESSES

Module PROCESSES declares the variables and data arrays that contain the information about the several simulation codes that are executed on the virtual machine as individual simulation tasks. These variables and data arrays constitute the ‘processes’ database described in Section 2.2.1.3.

## 3.14 Program PVMCATCHOUT

Program PVMCATCHOUT is an auxiliary program that is used as an intermediary in executing a simulation task on the virtual machine. It redirects the standard output file of the simulation task it initiates to a specified file.

## 3.15 Program PVMEXEC

Program PVMEXEC is the main program. It has been described in detail in Section 2.1.

## 3.16 Subroutine PVMFXREC

Subroutine PVMFXREC is the routine called to received a message from the several simulation tasks that are executing on the virtual machine. It divides the wait time into a number of wait intervals and calls the PVM library blocking routine ‘pvmftrecv’. If the blocking routine receives the specified message at any time during the wait interval, it returns with a error status of zero. PVMFXREC checks the errors status and returns if the error is zero. If the message has not been received during the wait interval, subroutine PVMFXREC checks to determine if a message with message tag 10002 has been received from any of the simulation tasks by a call to PVM library non blocking routine ‘pvmfnrecv’. This routine returns immediately regardless of whether the specified message has been received or not received. If such a message has been received, the error flag is set appropriately and the routine PVMFXREC returns to the calling routine. A message with message tag 10002 indicates that one of the simulation tasks has exceeded its wait time while waiting to receive a message from another simulation task. If a message with message tag 10002 has not been received, it checks to see if a message with message tag 10001 has been received from the virtual machine. If such a message has been received (denoting that one of the simulation tasks has terminated unexpectedly), it sets the error flag appropriately and returns to the calling routine. Finally, it checks to see if the number of wait intervals has been exceeded. If the number of wait interval has been exceeded, the error flag is set and routine PVMFXREC returns to the calling routine. If the number of wait intervals has not been exceeded, it returns to the top on the routine and begins the next wait cycle by calling the PVM library blocking routine ‘pvmftrecv’. The routine continue to cycle until either a message has been received or the number of wait cycles has been exhausted. The error flag values returns by subroutine PVMFXREC are listed in Appendix C.

## 3.17 Subroutine PVMGETFUNCS

Subroutine PVMGETFUNCS contains the source code for three C language auxiliary subroutines. These subroutine are getcwd, gethostname, and ignore_sigterm. Subroutine getcwd returns the name of the current working directory (the directory from which the PVMEXEC program was executed). Subroutine gethostname returns the name of the host on which the subroutine is executing. Subroutine ignore_sigterm tells the calling process to ignore the ‘SIGTERM’ signal from the operating system. Subroutine ignore_sigterm duplicates the functionality of subroutine ignsg15 and is not used by the PVMEXEC program.

## 3.18 Subroutine PVMOUTFILE

Subroutine PVMOUTFILE is a C language subroutine that serves as a wrapper around the PVM library subroutine ‘pvm_catchout’. Subroutine PVMCATCHOUT converts the input Fortran string variable into a null terminated C language string variable and passes it to subroutine ‘pvm_catchout’. This

subroutine is used by the PVMCATCHOUT program to redirect the standard output file of its child process to the file specified in the string variable.

## 3.19 Subroutine PVMSPAWNX

Subroutine PVMSPAWNX is a C language routine that serves as a wrapper around the PVM library function ‘pvm_spawn’. The routine converts the Fortran string variables in its input parameters into null terminated C string variables. The wrapper function was written because the Fortran interface for the ‘pvm-spawn’ PVM library routine does not allow command line parameters to be passed to the task that is being executed (spawned in PVM parlance).

## 3.20 Comdeck PVMVR5

Comdeck PVMVR5 is similar to the comdeck of the same name in the RELAP5-3D program. It defines the variables and parameters that indicate what types of coupling are being used by the coupled simulation currently being executed.

## 3.21 Module REPORT

Module REPORT declares the variables and data arrays used by subroutine POLL to determine if all simulation tasks have acknowledged the receipt of a message broadcast by the PVMEXEC program.

## 3.22 Comdeck TIMER

Comdeck TIMER declares the variables used to determine when the status line should be written to the output file of the PVMEXEC program and to the terminal screen.

## 3.23 Module TIMESTEPS

Module TIMESTEPS declares the variables and data arrays that contain the time step data. These variables and data arrays constitute the ‘timesteps’ database described in Section 2.2.1.5.

## 3.24 Subroutine TRAN

Subroutine TRAN control the transient simulation. It is described in great detail in Section 2.3.

## 3.25 Module UFILES

Module UFILES declares the unit numbers for the several files used by the PVMEXEC program.

# 4 References


1 W. L. Weaver, *The Application Programming Interface for the PVMEXEC Program and Associated Code Coupling System*, INL/EXT-05-00107, Idaho National Laboratory, March 2005.

2 C. Y. Paik and L. E. Hochreiter, *Analysis of FLECHT SEASET 163-Rod Blocked Bundle Data Using COBRA-TF*, NUREG/CR-4166, US Nuclear Regulatory Commission, 1986.

3 The RELAP5-3D© Code Development Team, *RELAP5-3D Code Manuals, Vol I, II, IV, and V*, INEEL-EXT-98-00834, Idaho National Engineering and Environmental Laboratory, Revision 2.2, October 2003.

4 Safety Code Development Group, *TRAC-PF1/MOD1: An Advanced Best-Estimate Computer Program for Pressurizer Water Reactor Thermal-Hydraulic Analysis*, LA-10157-MS, NUREG/CR-3858, Los Alamos National Laboratory, July 1986.

5 Fluent Corporation, *Fluent 6 User's Guide*, 2001.

6 J. W. Spore, et al., *TRAC-M/FORTRAN 90 (Version 3.0) Theory Manual*, NUREG/CR-6724, Los Alamos National Laboratory and Pennsylvania State University, July 2001.

7 K. K. Murata et al., *Code Manual for CONTAIN 2.0: A Computer Code for Nuclear Reactor Containment Analysis*, SAND97-1735, NUREG/CR-6533, Sandia National Laboratory, December 1997.

8 R. O. Gauntt et al., *MELCOR Computer Code Manuals: Version 1.8.5*, SAND2000-2417, NUREG/CR-6119, Sandia National Laboratory, October 2000.

9 P. J. Turinsky, R. M. K. Al-Chalabi, and P. Engrand, *NESTLE: A Few-Group Neutron Diffusion Equation Solver Utilizing the Nodal Expansion Method for Eigenvalue, Adjoint, Fixed Source Steady State and Transient Problems*, EGG-NRE-11406, Idaho National Engineering Laboratory, June 1994.

10 T. Downar et al., *PARCS: Purdue Advanced Reactor Core Simulator*, PU/NE-98-26, Purdue University, West Lafayette, IN, September 1998.

11 A. Geist et al., *PVM (Parallel Virtual Machine) User's Guide and Reference Manual*, ORNL/TM-12187, Oak Ridge National Laboratory, May 1993.

# Appendix A Message Tags

This appendix lists all of the message tags used by the PVMEXEC program along with a short description of the data in the message. Unless otherwise specified, the message specification is for the message that is sent from the PVMEXEC program to the designated simulation task. For these messages, the message acknowledgment is returned to the PVMEXEC in a message with the same message tag that contains a single data item in the message, the task identifier of the task sending the acknowledgement. Messages that are listed as pairs of send and receive messages with the same message tag define the handshaking for that message.

Message Tag 1

W1(R) Task transient type (-). A value of zero means a synchronous task and a value of 1 means an asynchronous task.

W2(R) Transient start time (s).

W3(R) Transient restart time (s)

W4(I) Not used (RELAP5-3D specific).

W5(I) Length of string containing simulation name (-).

W6-(A) Simulation name (-).

Message Tag 2

W1(I) Number of threads for multi-threaded simulation codes (-).

Message Tag 1000

W1(R) Global wait time (s).

W2(I) Debug flag (-).

Message Tag 1001

W1(I) Number of explicit send messages.

W2(I) Number of explicit receive messages.

Message Tag 1002

W1(I) Message tag (-).

W2(I) Task identifier of task receiving message (-).

W3(I) Message transient type (-). Zero means synchronous message, one means asynchronous message.

W4(I) Explicit coupling type (-). A value of -1 means parallel explicit coupling, a value of 0 means ‘leader’ in sequential explicit coupling, and a value of 1 means ‘follower’ in sequential explicit coupling.

W5(R) Wait time for acknowledgement (s).

W6(I) Length of character string defining thermal-hydraulic data to be sent for explicit coupling(-).

W7-(A) Character string defining thermal-hydraulic data to be sent for explicit coupling.

Message Tag 1003

W1(I) Message tag (-).

W2(I) Task identifier of task sending message (-).

W3(I) Message transient type (-). Zero means synchronous message, one means asynchronous message.

W4(I) Explicit coupling type (-). A value of -1 means parallel explicit coupling, a value of 0 means ‘leader’ in sequential explicit coupling, and a value of 1 means ‘follower’ in sequential explicit coupling.

W5(R) Wait time to receive message (s).

W6(I) Length of character string defining thermal-hydraulic data to be received for explicit coupling (-).

W7-(A) Character string defining thermal-hydraulic data to be received for explicit coupling.

Message tag 1004

W1(I) Number of send messages for semi-implicit coupling.

W2(I) Number of receive messages for semi-implicit coupling.

Message tag 1005

W1(I) Message tag (-).

W2(I) Task identifier of task receiving the message (-).

W3(R) Wait time for the message acknowledgment (s).

W4(I) Length of character string defining the thermal-hydraulic data to be sent for semi-implicit coupling (-).

W5-(A) Character string defining the thermal-hydraulic data to be sent for semi-implicit coupling.

Message tag 1006

W1(I) Message tag (-).

W2(I) Task identifier of task sending the message (-).

W3(R) Wait time to receive the message (s).

W4(I) Length of character string defining the thermal-hydraulic data to be received for semi-implicit coupling (-).

W5-(A) Character string defining the thermal-hydraulic data to be received for semi-implicit coupling(-).

Message tag 1007

W1(I) Number of send messages for kinetics coupling.

W2(I) Number of receive messages for kinetics coupling.

Message tag 1008

W1(I) Message tag (-).

W2(I) Task identifier of task receiving the kinetics message (-).

W3(R) Wait time to receive the message acknowledgment (s).

W4(I) Length of character string defining the kinetics data to be sent( -).

W5-(A) Character string defining the kinetics data to be sent.

Message tag 1009

W1(I) Message tag (-).

W2(I) Task identifier of task sending the kinetics message (-).

W3(R) Wait time to receive the message (s).

W4(I) Length of character string defining kinetics data to be received (-).

W5-(A) Character string defining kinetics data to be received.

Message tag 1010

W1(I) Number of send messages for control system coupling.

W2(I) Number of receive messages for control system coupling.

Message tag 1011

W1(I) Message tag (-).

W2(I) Task identifier of task receiving the control system message (-).

W3(R) Wait time to receive the message acknowledgment (s).

W4(I) Length of character string defining the control system data to be sent (-).

W5-(A) Character string defining the control system data to be sent.

Message tag 1012

W1(I) Message tag (-).

W2(I) Task identifier of task sending the control system message (-).

W3(R) Wait time to receive the control system message (s).

W4(I) Length of character string defining the control system data to be received (-).

W5-(A) Character string defining the control system data to be received.

Message tag 7000

W1(A) Fixed length string containing name of executable file for simulation task.

W2(A) Fixed length string containing command line parameters for simulation task.

W3(A) Fixed length string containing the name of the host in the virtual machine on which the execute the task.

W4(A) Fixed length string containing the name of the output file for the simulation task.

W5(I) Debugging flag (-).

Message tag 7001

W1(I) Task identifier of PVMEXEC task (-).

Message tag 8000

W1(R) Time for next minor edit (s).

W2(R) Time for next plot output (s).

W3(R) Time for next major edit (s).

W4(R) Time for next restart output (s).

W5(R) Time for next explicit data exchange (s).

W6(R) End time of transient (s).

W7(R) Maximum time step size for synchronously coupled tasks (s).

W8(R) Minimum time step size for synchronously coupled tasks (s).

Receive message tag 8001

W1(R) Desired time step size for synchronously coupled simulation task sending the message (s).

Send message tag 8001

W1(R) Global time step size for synchronously coupled simulation tasks (s).

Message tag 8002

W1(I) Task identifier of explicitly coupled task that is to send data to other explicitly coupled tasks for synchronous parallel explicit coupling. (s)

Message tag 8003

W1(I) Task identifier of explicitly coupled task that is to send data to other explicitly coupled tasks for asynchronous parallel explicit coupling (s).

Message tag 8004

W1(I) Task identifier of explicitly coupled task that is to send data to other explicitly coupled tasks for synchronous sequential explicit coupling (-).

Message tag 8005

W1(I) Task identifier of explicitly coupled task that is to send data to other explicitly coupled tasks for asynchronous sequential explicit coupling. (-)

Message tag 9000

W1(I) Task identifier of task sending message (-).

W2(I) Initialization status flag for task (-).

W3(I) Run status flag for task (-).

Message tag 9001

W1(I) Global initialization flag (-).

W2(I) Global run status flag (-).

Receive message tag 10000

W1(I) Task identifier of simulation task sending the message (-).

W2(I) Success flag for synchronously coupled task (-).

Send message tag 10000

W1(I) Global success flag (-).

Message tag 10001

W1(I) Task identifier of simulation code that has failed (-).

Message tag 10002

W1(I) Task identifier of simulation task that has exceeded its wait time for a message from another simulation task (-).

Message tag 10003

W1(I) Shutdown flag (-).

# Appendix B Data Dictionary

This appendix contains a listing of the significant local and global variables. Variables such as DO loop indices are not shown. The variables are listed alphabetically, along with where they are defined and with a short definition of the variable. The comdecks that were adapted from the comdeck of the same name in RELAP5-3D have some unused variables that should be removed in the future.

| Name | Module | Definition |
|---|---|---|
| aflag | contrl | Not used. |
| block | message | Not used. |
| clock_loop | contrl | Not used. |
| contrl_timestep | contrl | Index into proc_name array for task controlling the global time step. |
| curtex | dtstep | Time for next data exchange for explicit thermal-hydraulic coupling. |
| curtmi | dtstep | Time for next minor edit. |
| curtmj | dtstep | Time for next major edit. |
| curtpl | dtstep | Time for next plot output. |
| curtrs | dtstep | TIme for next restart output. |
| done | contrl | Status flag for transient advancements. |
| dt | contrl | Current time step size. |
| dtht | contrl | Not used. |
| dthy | contrl | Current time step size for thermal-hydraulic advancements. |
| exarch | machine | Architecture of machine executing the PVMEXEC program. |
| exflag | contrl | Flag for type of explicit thermal-hydraulic coupling. |
| explicit_coupling | contrl | Not used. |
| fail | contrl | Failure flag. |
| help | contrl | Not used. |

| | | |
|---|---|---|
| info | message | Error parameter returned by PVM library routines. |
| lines | inputd | Buffer to hold copy of input file as an array of strings. |
| machines | machine | Array of names of hosts in virtual machine. |
| march | machine | Array of architectures of hosts in virtual machine. |
| mdtid | machine | Task identifier of PVM daemon processes in the virtual machine. |
| mparms | machine | String containing PVM options for hosts in virtual machine. |
| msg_data | inputd | String variable defining data in message. |
| msg_extype | input | Type of explicit thermal-hydraulic coupling, i.e., parallel, ‘leader’ in sequential explicit coupling, or ‘follower’ in sequential explicit coupling. |
| msg_fromto | inputd | Task identifier for other task in message exchange. |
| msg_proc | inputd | Task identifier for first task in message exchange. |
| msgtag | message | Message tag for current message. |
| msg_mtype | inputd | Coupling type. Explicit thermal-hydraulic coupling, semi-implicit thermal-hydraulic coupling, kinetics coupling, or control system coupling. |
| msg_stype | inputd | Synchronous-asynchronous message flag. |
| msg_tag | inputd | Message tag for each message. |
| msg_type | inputd | Send-receive message flag. |
| msg_wait | inputd | Wait time for message. |
| mspeed | machine | Relative speed of the hosts in the virtual machine. |
| mydtid | pvmvr5 | Task identifier of PVM daemon process for PVMEXEC program. |
| mytid | pvmvr5 | Task identifier of PVMEXEC program. |
| narch | machine | Number of different architectures in the virtual machine. |
| nhost | machine | Number of hosts in the virtual machine. |

| | | |
|---|---|---|
| nmach | machine | Same as nhost. |
| ncast | message | Number of tasks to receive a broadcast message. |
| ncount | contrl | Number of attempted advancements. |
| nitem | message | Number of data items in a message. |
| nmechk | contrl | Flag to bypass mass error time step control. Set but not used. |
| nrepet | contrl | Counter for halving or doubling of the time step size. |
| nstsp | contrl | Not used. |
| num_msg | inputd | Total number of messages. |
| num_rcv | inputd | Number of receive messages. |
| num_snd | inputd | Number of send messages. |
| prevnd | dtstep | Time at end of previous time step interval. |
| print | contrl | Not used. |
| pfail | pvmexec | Global initialization failure flag. |
| proc_dtid | processes | Task identifier of PVM daemon process for each simulation task. |
| proc_dtype | processes | Data transmission mode for PVM daemon process for each simulation task. |
| proc_ifail | pvmexec | Initialization status flag for each task. |
| proc_mach | processes | Array of names of hosts for each simulation task. |
| proc_name | processes | Array of names of each task. |
| proc_xname | processes | Array of names of the executable file for each task. |
| proc_parms | processes | Array of strings containing the command lines parameters for each task. |
| proc_outfile | processes | Array of strings containing the file name for the standard output file for each task. |

| | | |
|---|---|---|
| proc_outtid | processes | Task identifier of PVMCATCHOUT task writing the redirected output file for each simulation task. |
| proc_run | pvmexec | Run status flag for each simulation task. |
| proc_stat | processes | Status flag for simulation tasks. |
| proc_stype | processes | Transient type for each simulation task. |
| proc_tid | processes | Task identifier for each simulation task. |
| proc_dtype | processes | Data transmission mode for message from the PVMEXEC program to each simulation task. |
| prun | pvmexec | Global run status. |
| pvmdatatype | message | PVM transmission mode for message (i.e., raw or xdr). |
| pvmdbg | pvmvr5 | Debug flag. |
| pvmdt | pvmvr5 | Global time step size. |
| pvmerr | pvmvr5 | Pvm error flag. |
| pvmexa | pvmvr5 | Control flag. True if explicit asynchronous explicit coupling used in coupled simulation. |
| pvmexs | pvmvr5 | Control flag. True if explicit synchronous explicit coupling used in coupled simulation. |
| pvmint4 | message | PVM data type for four byte integers. |
| pvmreal8 | message | PVM data type for eight byte real numbers. |
| pvmstrng | message | PVM data type for strings (arrays of characters). |
| pvm_smallerdt | contrl | Index into proc_name array for simulation task requesting a smaller time step size for a repeated advancement. |
| pvm_samedt | contrl | Index into proc_name array for simulation task requesting the same time step size for a repeated time step size. |
| pvm_stop | contrl | Index into proc_name array for simulation task wanting to terminate the couples simulation. |
| pvm_succes | contrl | Global success status flag. |

| | | |
|---|---|---|
| pvmsym | pvmvr5 | Control flag. True if synchronous coupling used in coupled simulation. |
| recvar | pvmvr5 | Receive buffer. |
| restart_time | inputd | Restart time for simulation. |
| semi-coupling | contrl | Not used. |
| simulation_name | inputd | String variable containing the simulation name. |
| skipt | contrl | Restart control flag. |
| sndtid | pvmvr5 | Task identifier of sending process. |
| sndvar | pvmvr5 | Send buffer. |
| start_time | inputd | Start time of simulation. |
| stride | message | Stride in memory for accessing data. Always equal to one. |
| sync | pvmvr5 | Control flag. True if task is a synchronous task. |
| tid | message | Task identifier. |
| timehy | contrl | Simulation time. |
| timeht | contrl | Not used. |
| timlft | dtstep | Time remaining in current time step interval. |
| wait | pvmvr5 | Global wait time. |

# Appendix C Error Codes

This appendix lists the error codes that are loaded into the pvmerr variable. The values are listed along with a short description of the error condition.

| Value | Description |
|---|---|
| -1 | A message with message tag 10003 has been received from the PVMEXEC program |
| -2 | An error occurred in subroutine pvmfxrec while waiting for a message. |
| -3 | An error occurred while testing for the receipt for a message with message tag 10003 from the PVMEXEC program |
| -4 | An error occurred while attempting to receive a message with message tag 10001. |
| -5 | Cannot determine my task identifier. |
| -mytid | The task has timed out waiting for a message from another simulation task where mytid is the task identifier of RELAp5-3D© |
| -extid | The PVMEXEC program has failed where extid is the task identifier of the PVMEXEC task |