SOFTWARE DESIGN AND IMPLEMENTATION DOCUMENT
FOR THE TRAC-M-SPECIFIC DATA MAP ROUTINE
IN THE COUPLED TRAC-M/PARCS CODE

R. Matthew Miller, Thomas J. Downar

School of Nuclear Engineering
Purdue University
W. Lafayette, IN 47907-1290

submitted to:

Division of Systems Technology
Office of Regulatory Research
Nuclear Regulatory Commission
Washington, D.C. 20555-001

Performed Under Subcontract to SCIENTECH, Inc.
Contract SC96-1026-001

June 8, 2000
I. Introduction

This document provides a description of the software design necessary to meet the Software Requirements Specification for the TRAC-M-Specific Data Map Routine (TDMR)\(^{(1)}\). The TRAC-M / TDMR code, which utilizes the FORTRAN-90 (F90) standard, is designed as a self-contained process which communicates with the General Interface process using standard message-passing constructs. Specifically, the Parallel Virtual Machine (PVM) package is utilized to handle all communication operations.

The TDMR functions as a secondary interface between the General Interface and TRAC-M. The software design description of Section II provides detail concerning the nature of this implementation within TRAC-M and outlines key components of the TDMR. Specifically, the variable, functional, and error-checking requirements are discussed, and the necessary coding is demonstrated. In addition, detail of the process control and I/O is presented.

II. Software Design

The TDMR is designed to maintain consistency with the requirements of both the General Interface\(^{(2)}\) and the PARCS-Specific Data Map Routine (PDMR)\(^{(3)}\), and the data to be transferred is specific to the TRAC-M/PARCS coupling. In this implementation, entry into the functional units of the TDMR is controlled by TRAC-M through subroutine calls. In addition, the transfer of data between TRAC-M and the TDMR is achieved through the use of the TRAC-M module constructs.

The TDMR contains three functional units and one error-checking unit. The first functional unit initializes the PVM process and establishes communication with the General Interface process and manages the communication of initial control information with the General Interface process. The second and third functional units handle both the transfer of time-dependent control information and the mapping of property data between the General Interface and PARCS processes. Finally, in the error-checking unit, tests are performed to ensure that the information sent both to and from the General Interface is accurate.

The organization of the subroutines takes advantage of the F90 MODULE construct. As will be shown in the following subsections, modules are used to group subroutines with similar application. Specifically, the subroutines associated with the initialization and contained within module TDMRInitCalc; the subroutines associated with the time-dependent data mappings are contained within module TDMRTimeCalc; the subroutines associated with the communication of control buffers between processes are contained within module TDMRCommBuf; and the subroutines associated with the error-checking are contained within module TDMRErrorCheck. A brief description of the subroutines and modules is provided in Appendix B.

II.A. Variable Definition
The scalar and array variables used by the TDMR are grouped into seven separate classes, and a complete description of these variables, which includes data type, dimension, and range of values, is provided in Appendix A. The first class, shown in Table 1, relates to the variables needed for process control with PVM. The second class involves the neutronic vector and control buffer data sent by the General Interface (i.e., sent from PARCS), and is described in Table 2. The third class, described in Table 3, involves the thermal-hydraulic vector and control buffer data sent to the General Interface (i.e., sent to PARCS). The vectors in the second and third classes, which contain the space-dependent property data, will utilize SI units and have the structure shown in the SRS for the TDMR (1).

The fourth class of variables, shown in Table 4, relate to the parameters and dimension variables used for the vectors of property data. Table 5 describes miscellaneous calculation control variables. Finally, Table 6 describes the variables associated with error-checking.

The variables shown in Tables 1-6 are declared in module TDMRVarDecl, which is used by each subroutine in the TDMR. The declaration of all the variable arrays described in Tables 1-6 takes advantage of dynamic memory allocation of F90, thus static dimension parameters are not required.

A single header file, fpvm3.h, is needed in the TDMR routines for PVM functionality. Only one header file from TRAC-M is needed by the TDMR in order to function properly. Access to TRAC-M memory locations requires use of several modules which contain the derived data types TRAC-M uses for storing its data. A description of the TRAC-M variables in these modules which are used by the TDMR is provided in Table 7. It should be noted that some of these modules are used only for the subroutines which are contained in them.

```plaintext
INCLUDE constant
MODULE BadInput
MODULE BreakArray
MODULE BreakVlt
MODULE ChanVlt
MODULE CompTyp
MODULE CorePowerAr
MODULE CorePowerDat
MODULE CSDat
MODULE CSEval
MODULE FillArray
MODULE FillVlt
MODULE Flt
MODULE Gen1DArray
MODULE Global
MODULE GlobalDat
MODULE GlobalDim
MODULE HSAarray
MODULE IntrType
MODULE Io
MODULE PipeVlt
```
II.B. TRAC-M Process Control

The process flow for TRAC-M when coupled to the General Interface, and the entry points into the TDMR from TRAC-M, is shown in Figure 1. As mentioned, the TDMR contains three functional units, corresponding to initialization, thermal-hydraulic/heat structure to neutronic mapping, and neutronic to thermal-hydraulic/heat structure mapping, which is denoted in the figure by TDMR(1), TDMR(2), and TDMR(3), respectively. Entry into these units is made from trac.f90, steady.f90, and trans.f90, and is controlled using an integer route identifier (iroute) which has values of 1, 2, or 3, and is passed as a subroutine argument to the TDMR. The following lines of code depict the routing logic used in subroutine TDMR(). The code section shown below also demonstrates part of the exit handling procedure which will be discussed in more detail in Section II.G.

```
CheckRoute: SELECT CASE (iroute)
CASE(1)
   CALL TDMRInit()
   IF (errtdmr) THEN
      CALL TDMRClean()
      CALL TDMRExit()
   ENDIF
CASE(2)
   CALL TDMRth2n()
   IF (tdmrdone .OR. errtdmr) THEN
      CALL TDMRClean()
      CALL TDMRExit()
   ENDIF
CASE(3)
   CALL TDMRn2th()
   IF (errtdmr) THEN
      CALL TDMRClean()
      CALL TDMRExit()
   ENDIF
END SELECT CheckRoute
```

The process flow through the three functional units is depicted in Figure 2, and Sections II.C through II.E will discuss in more detail the calculational procedure for each. In addition, a brief description of the subroutines used by the TDMR is provided in Appendix B.
The first step in the TRAC-M procedure is to process the user input. A single input option specifies the coupling of TRAC-M to a 3-D kinetics code. An integer namelist variable, \texttt{itdmr}, has been created for the purpose of using the kinetics code. If \texttt{itdmr} is not present or is set to 0 in the \texttt{NAMELIST} assignments, TRAC-M will run in stand-alone mode, skipping over all calls to the TDMR and using its internal point kinetics routine, if appropriate. If \texttt{itdmr} is assigned a value of 1, TRAC-M assumes that the current calculation is to be coupled with a 3-D or 1-D kinetics calculation. An integer \texttt{NAMELIST} variable, \texttt{itmrp} was also created to switch on or off the verbosity with which the TDMR will operate. Defining this variable equal to 1 will cause all TDMR output to be written to the screen, the TRAC-M output file, \texttt{TRCOUT}, and the TRAC-M

Figure 1: Flow Control Through TRAC-M / TDMR
message file, TRCMSG. Omitting this variable from NAMELIST definitions or defining it to be 0 supresses nearly all TDMR output.

Once the TRAC-M/TDMR initialization process is complete, the second unit of the TDMR is called to pass the initial thermal-hydraulic and heat structure data to PARCS. PARCS then calculates an initial steady-state eigenvalue, keeping total core power constant. The initial power shape is then received from PARCS via the General Interface with a call to the third unit of the TDMR.

Steady-state calculations begin when the program trac.f90 makes the subroutine call to steady.f90. Steady-state calculations in TRAC-M are determined based on the input and involve performing a pseudo-transient. Specifically, time is advanced in the same way as a transient calculation, but total core power is held constant throughout the steady-state calculation, and the time step size is artificially increased to accelerate convergence. TRAC-M determines steady-state convergence by comparing several thermal-hydraulic parameters against the convergence criterion. This criterion has a default value of $1.0e^{-04}$, although it may be user-specified in the TRAC-M input deck. During the steady-state calculation, calls are made to the second and third units of the TDMR to obtain the new power shape after the thermal-hydraulic solution has been advanced in time. Once the steady-state calculation has converged, the calculation exits here after
a “stutter” step has been performed. The stutter step involves nothing more than sending PARCS a logical variable defining whether the calculation to be terminated. Figure 3 shows calculational flow for a coupled steady-state initialization calculation.

![Diagram of calculational flow](image)

**Figure 3: Calculational Flow for Coupled Steady-State Calculation**

Transient calculations begin when the program `trac.f90` makes the subroutine call to `trans.f90`. This subroutine begins by calculating the thermal-hydraulic solution for the new time step. After the new thermal-hydraulic solution has been obtained, calls are made to the second and third unit of the TDMR to obtain the power deposited in both the fuel and the coolant at the new time step. It should be noted that the steady-state power distribution (more specifically,
the power distribution received in the initial call to the third unit of the TDMR) is used to calculate the thermal-hydraulic solution of the first transient time step. This process is repeated until either the end of problem time is reached or an error signal is received. Figure 4 shows calculational flow for a coupled transient calculation. A more detailed description of the exit handling procedure is given in Section II.G.

![Diagram](image)

**TRAC-M Initialization**

- TDMR(1)
- TDMR(2)
- TDMR(3)

Obtain first eigenvalue solution from PARCS

**IF NOT FINISHED**

- prep
- hout
- post

**SUBROUTINE TRANS**

- TRAC-M performing actual transient calculation
- PARCS solving transient fixed-source problem

- $\text{tdmrdone} = .\text{TRUE}.$

- pass logical variable to PARCS

**TDNR & TRAC-M Exit**

Figure 4: Calculational Flow for Coupled Transient Calculation

It should be noted that restart problems can be performed with the coupled TRAC-M/PARCS code. In order for a restart calculation to be performed, both TRAC-M and PARCS need to read their respective restart files. A calculation can be restarted from steady-state to begin a
transient calculation based on a steady-state condition, or a transient calculation can be restarted in order to continue the previously stopped transient calculation. Figure 5 shows the process flow for restarting a transient calculation from a steady-state condition, while Figure 6 shows the same for a transient calculation restarted from a transient condition.

Figure 5: Process Flow for Transient Restart from Steady-State Condition
Figure 6: Process Flow for Transient Restart from Transient Condition
Due to the ability of the coupled code to perform restart calculations, restart edits in the coupled TRAC-M/PARCS code are synchronized. The calculation loop within subroutine steady and tran is unrolled in Figure 7, which demonstrates the synchronization of the restart edits.

```
timet = 1.5
delt = 0.1
rsted = .FALSE.
DO WHILE ...
  prep
  hout
  post
ENDDO
```

**Figure 7: Synchronization of Restart Edits in the Coupled TRAC-M/PARCS Code**
II.C. Initialization

The initialization unit control two basic tasks: (1) communicate process IDs with the General Interface, and (2) communicate initial control buffers with the General Interface. These tasks are performed by subroutines contained in the module `TDMRInitCalc`.

Task (1) is performed by subroutine `TDMRObtainIDs()`, and involves first enrolling the TRAC-M process in PVM by calling the function `pvmfmytid()` to obtain the TRAC-M/PVM process ID. The TRAC-M process will then join a dynamic process group consisting of the PARCS and General Interface processes, and wait on a barrier for these processes. Once all of the processes have arrived at the barrier, the TDMR will receive the process ID broadcasted from the General Interface, and send the TRAC-M process ID to the General Interface. This procedure provides communication coherency between the General Interface and TRAC-M processes, and is performed by using pre-determined message tags (`mtypegi`, `mtypeth`), which are consistent with those used by the General Interface\(^4\). The following lines of code demonstrate the necessary procedure:

```fortran
! Establish communication with the General Interface process.
ntasks = 3
mtypegi = 1
mtypeth = 2
group = 'procs'
! Enroll the TRAC-M process in PVM.
CALL pvmfmytid( tidth)
! Join the dynamic process group and wait for the General Interface
! and neutronic processes.
CALL pvmfjoingroup( group, inum)
CALL pvmfbarrier( group, ntasks, info)
! Receive ID broadcasted from the General Interface process.
CALL pvmftrecv( -1, mtypegi, timeout, 0, info)
CALL pvmfunpack( INTEGER4, tidgi, 1, 1, info)
! Send ID to the General Interface process.
CALL pvmfinitsend( PVMDEFAULT, info)
CALL pvmfpack( INTEGER4, tidth, 1, 1, info)
CALL pvmfsend( tidgi, mtypeth, info)
```

It should be noted that the function `pvmftrecv`, which was shown above and is used throughout the code, is a non-blocking receive which terminates if no message has arrived after `timeout` seconds. The variable `timeout` is described in Table 1 of Appendix A and is set to a value of 600 seconds.

Task (2) is controlled by subroutine `TDMRBufInit()`. The first step of task (2) is to receive and unpack the initial PARCS control buffers. The buffers, along with the buffer dimensions, are sent from the General Interface as one structure. This data structure is received and unpacked in subroutine `TDMRRecvBufn()` . The data to be received in the control buffers is described in the SRS for the TDMR, and the procedure required to unpack the buffers is depicted in the following lines of code. The variables copied from the control buffers are described in the tables of Appendix A, and are consistent with the data being sent by the PARCS-Specific Data Map Routine.
! Receive data structure from the General Interface process.
CALL pvmfrecv( tidgi, mtypegi, timeout, 0, info)
    istride = 1
! Extract buffer dimensions and allocate memory for the
! PARCS control buffers.
CALL pvmfunpack( INTEGER4, dimbuf, 6, istride, info)
    ALLOCATE( cbufn(dimbuf(1)))
    ALLOCATE( lbufn(dimbuf(2)))
    ALLOCATE( i2bufn(dimbuf(3)))
    ALLOCATE( i4bufn(dimbuf(4)))
    ALLOCATE( r4bufn(dimbuf(5)))
    ALLOCATE( r8bufn(dimbuf(6)))
! Extract data type-dependent control buffers.
CALL pvmfunpack( STRING, cbufn, 6*dimbuf(1), istride, info)
CALL pvmfunpack( INTEGER4, lbufn, dimbuf(2), istride, info)
CALL pvmfunpack( INTEGER2, i2bufn, dimbuf(3), istride, info)
CALL pvmfunpack( INTEGER4, i4bufn, dimbuf(4), istride, info)
CALL pvmfunpack( REAL4, r4bufn, dimbuf(5), istride, info)
CALL pvmfunpack( REAL8, r8bufn, dimbuf(6), istride, info)

Upon return from subroutine TDMRRecvBufn(), the necessary data are extracted from the
buffers, and the memory for the buffers is deallocated, as shown below. It should be noted that in
the case where PARCS is not sending the permutation matrices, it is assumed that the automatic
mapping functionality in the General Interface is to be used. In this case, some of the data nor-
mally sent by PARCS in the Initial Neutronic Control Buffer will not be present. These missing
data are inferred from the TRAC-M database of components stored in memory.

! Extract data from the PARCS control buffers.
errcalcn = lbufn(1)
errdatan = lbufn(2)
errpvmn = lbufn(3)
errdatagi = lbufn(4)
errpvmgi = lbufn(5)
pdmrdone = lbufn(6)
recvn = lbufn(7)
cbccalc = lbufn(8)
crcntl = lbufn(9)
итрип = i4bufn(1)
IF (recvn) THEN
    numvols = i4bufn(2)
    numhss = i4bufn(3)
    ncolth2n = i4bufn(4)
    nrown2th = i4bufn(5)
ENDIF
ncrcntl = i4bufn(6)
IF (.NOT.crcntl) ncrcntl = 0
IF (ncrcntl .GT. 0) THEN
    ALLOCATE( isgv(ncrcntl))
    ALLOCATE( sgvbank(ncrcntl))
    DO i=1,ncrcntl
        isgv(i) = i4bufn(6+2*i-1)
        isgvbank(i) = i4bufn(6+2*i)
        initcrp(i) = r8bufn(i)
Once these data have been copied to the appropriate TDMR memory locations, the mapping data in the vectors ivolndx and ihsndx are decoded and checked for accuracy. The trip ID number is checked for existence, and if a critical boron concentration calculation is indicated, a check is performed to verify whether solute tracking is activated in TRAC-M. An additional test is performed to verify that the initial boron concentration in all cells defined in %TABLE1 of MAPTAB is nonzero. The error-checking procedure will be discussed in Section II.F. After the mapping information has been checked for errors, the initial TRAC-M control buffer is constructed. The following lines of code depict the necessary procedure, which is performed in subroutine TDMRBufInit(), and the variables used below are described in the tables of Appendix A. The geometry descriptions of the heat structures defined in the mapping information must be passed to PARCS in the initial TRAC-M buffer. The PDMR uses these descriptions for determination of the method used for calculating Doppler temperature. If the heat structure is of Cartesian geometry, the TDMR will send the string “SLAB” in the data type-dependent control buffer. If the heat structure is of cylindrical geometry, the TDMR will send “PIN.” The geometry descriptors are not passed to PARCS if the MAPTAB file is not present.
recvth = .FALSE.
IF (.NOT.recvn) THEN
    automap = .TRUE.
ENDIF

! Check whether boron exists in the system and solute tracking
! is turned on if a critical boron calculation has been specified
! by PARCS. If no solute exists, turn the calculation off.
IF (ssinit .AND. cbccalc .AND. .NOT.errdatath) THEN
    ! Check to see if solute tracking is turned on.
    IF (isolut) THEN
        ! Check to see whether a non-zero boron concentration initially exists in
        ! the TRAC-M system.
        DO m = 1,numvols
            IF (is3d(m)) THEN
                IF (vessTab(compptrs(m))%iconc .NE. 0) THEN
                    IF (vsAr3(compptrs(m))%conc(iptrs(m),
                        &                     jptrs(m),kptrs(m)) .GT. 0.0D+00)  GOTO 10
                ENDIF
            ELSE
                IF (pipeTab(compptrs(m))%iconc .NE. 0) THEN
                    IF (g1DAr(compptrs(m))%concn(iptrs(m))
                        &                     .GT. 0.0D+00)  GOTO 10
                ENDIF
            ENDDO
            cbccalc = .FALSE.
        ELSE
            cbccalc = .FALSE.
       ENDIF
    ENDIF
10   CONTINUE
ENDIF

! Pack the control buffers and start time.
lbufth(1) = errcalcth
lbufth(2) = errdatath
lbufth(3) = errpvmth
lbufth(4) = errdatagi
lbufth(5) = errpvmgi
lbufth(6) = tdmrdone
lbufth(7) = recvth
lbufth(8) = rstrt
lbufth(9) = ssinit
lbufth(10) = cbccalc
lbufth(11) = crcntl
r8bufth(1) = timet

! Pack the Heat Structure geometry descriptions.
IF (.NOT.errdatath .AND. .NOT.automap) THEN
    DO i = 1,numhss
        IF (isslab(i)) THEN
            cbufth(i) = 'SLAB'
        ELSE
            cbufth(i) = 'PIN'
        ENDIF
    ENDDO
ENDIF
It should be noted that the value of the variable rstrt is initially .FALSE., but is set to .TRUE. when TRAC-M opens TRCRST. Once these data have been copied, subroutine TDMRSendBufth() is called to pack these control buffers, along with the dimension of each buffer, into a single PVM data structure and then send the structure to the General Interface process. The procedure is performed as follows:

```fortran
istride = 1
CALL pvmfinitsend( PVMDEFAULT, info)
! Pack buffer dimensions.
CALL pvmfpack( INTEGER4, dimbuf, 6, istride, info)
! Pack data type-dependent control buffers.
CALL pvmfpack( STRING, cbuth, 6*dimbuf(1), istride, info)
CALL pvmfpack( INTEGER4, ibuth, dimbuf(2), istride, info)
CALL pvmfpack( INTEGER2, ibuth, dimbuf(2), istride, info)
CALL pvmfpack( REAL4, r4buth, dimbuf(5), istride, info)
CALL pvmfpack( REAL8, r8buth, dimbuf(6), istride, info)
! Send data structure to General Interface process.
CALL pvmfsend( tidgi, mtypeth, info)
```

Once the data structure has been sent, the memory for the buffers can be deallocated as shown below:

```fortran
! Deallocate memory for the TRAC-M control buffers.
DEALLOCATE( cbuth)
DEALLOCATE( ibuth)
DEALLOCATE( ibuth)
DEALLOCATE( ibuth)
DEALLOCATE( r4buth)
DEALLOCATE( r8buth)
```

In order for the General Interface to construct the permutation matrices necessary for mapping thermal-hydraulic and heat structure data to the neutronics domain and vice versa, data from the TRAC-M component database must be passed to the General Interface. The message passing occurs after the Initial Thermal-Hydraulic Control Buffer has been sent, thus creating a “secondary” Initial Thermal-Hydraulic Control Buffer. The coding performing the retrieval of these data is in subroutine TDMRBufInit() in module TDMRInitCalc and is as follows:

```fortran
IF (automap) THEN
! Allocate memory for temporary variables used for extracting
! the CHAN component information from the TRAC-M database.
ALLOCATE( temp1(nchans), temp2(nchans), temp3(nchans)
 &
 temp4(nchans), temp5(nchans))
! Compute the number of cells and the number of axial heat
! structure levels coupled with the neutronics calculation.
! Also store the component numbers and cco values of the CHAN
! components encountered.
! temp1(:) contains the CHAN component numbers, unsorted
! temp3(:) contains the CHAN cco values, unsorted
DO ii = 1,nchans
 tcp = cPowerAr%ccochan(ii)
```
temp1(ii) = genTab(tcp)%num
temp3(ii) = tcp

temp5(ii) = INT(hsAr(tcp+1)%hcelo(rodTab(tcp+1)%ncrz+1)) -
& INT(hsAr(tcp+1)%hcelo(2)) + 1
numvols = numvols + temp5(ii)
umhss = numhss + (rodTab(tcp+1)%ncrx * (rodTab(tcp+1)%ncrz+1))

ENDDO

After this block of code is executed, temp1(:) contains the unique component numbers of all the CHAN components in the TRAC-M model. The derived data type array element number for each CHAN component is contained in temp3(:), and the number of thermal-hydraulic cells in each CHAN component that are coupled with neutronics is contained in temp5(:). The component numbers and array element vectors are not sorted. Sorting from lowest component number to highest component number is important for consistency in the automatic mapping. The amount of data which needs to be passed to the General Interface is computed, and the sorting of the component numbers is performed as follows:

! Calculate the correct dimensions for the outgoing and incoming!
! time-dependent vectors. These are important for error-checking.
ncolth2n = (nsubth2n * numvols) + (nsubhs2n * numhss)
nrown2th = (nsubn2th * numvols) + (nsubn2hs * numhss)
! Sort the component numbers from lowest to highest, and use the same
! swap scheme for the cco pointers. We can then send the T/H data in
! the exact same manner as is done with regularly-mapped T/H.
DO ii = 1,nchans
    next = 1
    comp = 10000
    DO jj = 1,nchans
        IF (temp1(jj) .LE. comp) THEN
            next = jj
            comp = temp1(jj)
        ENDIF
    ENDDO
    temp2(ii) = temp1(next)
    temp4(ii) = temp3(next)
    temp1(next) = 10000
ENDDO
! temp2(:) contains the CHAN component numbers, sorted
! temp4(:) contains the CHAN cco values, sorted
! temp5(:) contains the number of cells coupled, sorted

Now the composite ID numbers for both thermal-hydraulic and heat structure components need to be constructed based on the data found in TRAC-M memory. The composite ID numbers are explained in Section II.A.1 of the SRS for the TDMR. Construction of the composite ID numbers is shown below:

! Construct the ivolndx(:) vector as if it came from the neutronics!
! code. This can be done since for 1-D kinetics, we know all the CHAN!
! components will be mapped to kinetics. After constructing this!
! vector, we will send it through the same machinery as normally!
! mapped data for the automatic computation of the other pointer
! parameters.
ALLOCATE( ivolndx(numvols))
m = 1
DO ii = 1,nchans
tcp = temp4(ii)
kbot = INT( hsAr(tcp+1)%hcelo(2))
top = INT( hsAr(tcp+1)%hcelo(rodTab(tcp+1)%ncrz+1))
DO jj = kbot,top
   ivolndx(m) = temp2(ii) * 1000000 + jj*1000
   m = m + 1
ENDDO
ENDDO

! Construct the ihsndx(:) vector as if it came from the neutronics code. This can be done since for 1-D kinetics, we know all the CHAN components will be mapped to kinetics. After constructing this vector, we will send it through the same machinery as normally mapped data for the automatic computation of the other pointer parameters. We want the ROD to have the same component number as the T/H part of the CHAN component in order to take advantage of the coding already done for CHANs.

! The cco value of a CHAN ROD is the cco value of the CHAN itself + 1.
ALLOCATE( ihsndx(numhss))
m = 1
changroups = 0
chanlevels = 0
DO ii = 1,nchans
tcp = temp4(ii)
chanlevels = chanlevels + rodTab(tcp)%ncrz
DO jj = 1,rodTab(tcp)%ncrx
   changroups = changroups + 1
   DO kk = 1,rodTab(tcp)%ncrz + 1
      ihsndx(m) = temp2(ii) * 1000000 + jj * 1000 + kk
      m = m + 1
   ENDDO
ENDDO
ENDDO

! After all these steps, TRAC-M doesn’t know whether the data came from the neutronics code or if it constructed it. Therefore, we can call all the relevant error-checking routines to make sure this algorithm is correct, as well as allowing the error-checking routine to calculate the necessary pointers. The call to case 2 checks the T/H cells, while the call to case 3 checks the heat structure mapping.
CALL TDMRDataErrchk(2)
CALL TDMRDataErrchk(3)

At this point, all the data required by the General Interface for constructing the permutation matrices has been found in TRAC-M memory and checked for accuracy and consistency. It is safe to pass these data to the General Interface now, which is performed according to the coding below:

! Since error-checking has been performed on the data, we can now safely pass these data to the GI along with logicals indicating
! whether these data are valid.
  dimbuf = (/0, 3, 0, 1+5*nchans, 0,
            & numvol+2*changroups+chanlevels/)

! Pack the error logicals first, and simply skip over the rest if
! there has been an error.
  lbufth(1) = errcalcth
  lbufth(2) = errdatath
  lbufth(3) = errpvmth
  IF (errcalcth .OR. errdatath .OR. errpvmth)  GOTO 999

! Now we loop over the CHAN components in the system and pack the data
! as needed.
  i = 1
  m = 2
  i4bufth(1) = nchans
  DO ii = 1,nchans
    i4bufth(m) = temp2(ii)
    i4bufth(m+1) = temp5(ii)
    i4bufth(m+2) = chanTab(temp4(ii))%icrnk
    DO jj = 1,temp5(ii)
      r8bufth(i) = g1DAr(temp4(ii))%dx(jj+i4bufth(m+2))
      i = i + 1
    ENDDO
    m = m + 3
  ENDDO

! All the important CHAN T/H data are packed. Next are the data
! for the heat structure (ROD) parts of the CHAN components in the
! model.
  DO ii = 1,nchans
    i4bufth(m) = rodTab(temp4(ii)+1)%ncrx
    i4bufth(m+1) = rodTab(temp4(ii)+1)%ncrz
    m = m + 2
    DO jj = 1,rodTab(temp4(ii)+1)%ncrx
      r8bufth(i) = hsAr(temp4(ii)+1)%cpowr(jj)
      r8bufth(i+1) = hsAr(temp4(ii)+1)%nrdx(jj)
      i = i + 2
    ENDDO
    DO jj = 1,rodTab(temp4(ii)+1)%ncrz
      r8bufth(i) = hsAr(temp4(ii)+1)%rdz(jj)
      i = i + 1
    ENDDO
  ENDDO

! Send all these buffers to the General Interface.
  CALL TDMRSendBufth()

Once these data are sent to the General Interface, the GI will process them and perform further checks for consistency with similar data sent from PARCS. The GI will then broadcast an additional buffer to both TRAC-M and PARCS, stating whether the current calculation is ready to begin (error-free) or whether an error has occured. Coding for handling the communication with the GI is shown below:

! As long as things went fine up to this point, the GI will
! echo back its error status. First check the error status of
! the current run.
II.D. TRAC-M to General Interface Mapping

This functional unit performs two basic tasks: (1) process the TRAC-M time-dependent control buffers, and (2) send the buffers and the unpermuted vector of thermal-hydraulic and heat structure data. The two subroutines utilized for these tasks, TDMRBufth2n() and TDMRMapth2n(), respectively, are contained in module TDMRTimeCalc. The procedure of TDMRBufth2n() is similar to that used in TDMRBufInit() for the sending of the TRAC-M initial control buffers. The difference here is seen in the data placed in the buffers, as shown below. In addition, these buffers are not sent immediately, but rather, are sent prior to the sending of the vector of thermal-hydraulic data in TDMRMapth2n(). This is done so that a data or PVM error occurring during the packing of the vector of thermal-hydraulic data can still be communicated to the other processes. The data in the buffers are described in the SRS for the TDMR, and a description of the variables is given in the tables of Appendix A.

IF (errcalc.th .OR. errdatath .OR. errpvmth) GOTO 999

! Receive GI status buffer.
CALL TDMRRecvBufn()
IF (errpvmth) GOTO 999

! Extract data from the initial GI buffer.
errcalcgi = lbufn(1)
errdatagi = lbufn(2)
errpvmgi = lbufn(3)

999 CONTINUE

! Allocate memory for the TRAC-M control buffers.
dimbuf = (/0, 9, 0, 0, 0, 1+ncrcntl/)
ALLOCATE( cbufth(dimbuf(1))
ALLOCATE( lbufth(dimbuf(2))
ALLOCATE( i2bufth(dimbuf(3))
ALLOCATE( i4bufth(dimbuf(4))
ALLOCATE( r4bufth(dimbuf(5))
ALLOCATE( r8bufth(dimbuf(6))

! Pack the TRAC-M control buffers.
lbufth(1) = errcalc.th
lbufth(4) = errdatagi
lbufth(5) = errpvmgi
lbufth(6) = tdmrdone
lbufth(7) = rsted
lbufth(8) = majed
r8bufth(1) = delt

IF (ncrcntl) THEN
  DO ii = 1,ncrcntl
    DO jj = 1, csGl%ntsv
      IF (IABS(csSig(jj)%idsv) .EQ. isgv(ii))  GOTO 100
    ENDDO
    CALL error(1, '*TDMR* control rod position mapping error')
  RETURN
  100 CONTINUE
ENDDO

IF (csSig(jj)%presVal.ge.1.0_sdk.OR.csSig(jj)%presVal.le.0.0) THEN
  CALL error(1, '*TDMR* control rod position is >1.0 or <0.0')
RETURN
ENDIF
newcrp(ii) = csSig(ii)%presVal
r8bufth(1+ii) = newcrp(ii)
ENDDO
ENDIF

! Reset rsted and majed after packing into control buffer.
  rsted = .FALSE.
  majed = .FALSE.
!! Check the status of the trip card specified
!! in the mapping input file, MAPTAB.
IF (.NOT.scramtrip) THEN
  IF (itrip .NE. -10000) THEN
    CALL trip( itrip, iset, dum1, dum2)
    IF (iset .NE. 0) THEN
      scramtrip = .TRUE.
    ELSE
      scramtrip = .FALSE.
    ENDIF
  ELSE
    scramtrip = .FALSE.
  ENDIF
ELSE
  scramtrip = .FALSE.
ENDIF
ENDIF
lbufth(9) = scramtrip

The second and third words of the logical control buffer will be assigned in TDMRMapth2n() following the data error-checking and prior to the sending of the control buffers.

The first step of Task (2) is to allocate space for the vector of thermal-hydraulic and heat structure data to be sent to the General Interface, the coding for which is shown below from subroutine TDMRMapth2n() in module TDMRTimeCalc:

! Copy data from TRAC-M memory to vector buffer.
  nvecth = nsubth2n*numvols + nsubhs2n*numhss
  IF (.NOT.ALLOCATED(vecth))  ALLOCATE( vecth(nvecth))

As described in the SRS for the TDMR, the thermal-hydraulic and heat structure data to be sent to PARCS involves (in order): moderator temperature (K), moderator liquid density (kg/m³), moderator vapor density (kg/m³), moderator void fraction, Boron concentration (ppm), average fuel temperature (K), fuel centerline temperature (K), and fuel surface temperature (K). The memory locations of these data are determined in the error-checking routines called from subroutine TDMRInit(). Pseudo-pointers to the correct memory locations are stored during this error-checking and used here for the expedient packing of the thermal-hydraulic and heat structure data. The following variables are used as pseudo-pointers and are listed in Table 3 of Appendix A: compptrs(:), iptrs(:), jptrs(:), kptrs(:), hsptrs(:), hsrods(:), hsaxial(:), hsfsn(:). The necessary coding is shown below, and the variables used are described in the tables of Appendix A. It should be noted that within TRAC-M, a different memory structure is used for 3-D components and 1-D components. Therefore, the dimensionality of each thermal-hydraulic component must be determined before packing its data so that the correct array will be accessed. The dimensionality of each component is determined in the error-checking procedure.
built into the decomposition of the composite ID numbers. Each thermal-hydraulic cell coupled to the kinetics calculation has an entry in the logical \texttt{is3d(,)} array, which indicates whether the cell is contained within a VESSEL component. In the same manner as \texttt{is3d(,)}, each thermal-hydraulic cell coupled to the kinetics calculation has an entry in the logical \texttt{ischan(,)} array, which indicates whether the cell is contained within the thermal-hydraulics portion of a CHAN component. The arrays \texttt{vsAr3} and \texttt{g1DAr} are for three-dimensional (VESSEL) components and one-dimensional (e.g., PIPE/CHAN) components, respectively. These arrays are derived data types, and are declared within TRAC-M modules \texttt{VessArray3} and \texttt{Gen1DArray}, respectively. These modules must be \texttt{USE}d by \texttt{TDMRMaph2n()} in order to access these arrays. Similarly, most heat structure data is stored in derived data type \texttt{hsAr}, which is declared in TRAC-M module \texttt{HSArray}. In order to access the heat structure data, module \texttt{HSArray} must be \texttt{USE}d by \texttt{TDMRMaph2n()}. The calculation of the average fuel temperature depends on the geometry of the heat structure, e.g., ROD or SLAB. For this reason, the geometry of each heat structure axial level coupled to the kinetics calculation is stored in the logical array \texttt{isslab(,)} when the heat structure composite ID numbers were decoded after receipt through the initial neutronic control buffer (or automatic construction during the TDMR initialization phase).

! Pack thermal-hydraulic and heat structure vector.
  l = 0
! Moderator temperature:
  DO i = 1,numvols
    l = l + 1
    tcp = compptrs(i)
    IF (is3d(i)) THEN
      vecth(l) = vsAr3(tcp)%tln(iptrs(i),jptrs(i),kptrs(i))
    ELSE
      vecth(l) = g1DAr(tcp)%tln(iptrs(i))
    ENDIF
  ENDDO
! Moderator Liquid Density:
  DO i = 1,numvols
    l = l + 1
    tcp = compptrs(i)
    IF (is3d(i)) THEN
      vecth(l) = vsAr3(tcp)%roln(iptrs(i),jptrs(i),kptrs(i))
    ELSE
      vecth(l) = g1DAr(tcp)%roln(iptrs(i))
    ENDIF
  ENDDO
! Moderator Vapor Density:
  DO i = 1,numvols
    l = l + 1
    tcp = compptrs(i)
    IF (is3d(i)) THEN
      vecth(l) = vsAr3(tcp)%rovn(iptrs(i),jptrs(i),kptrs(i))
    ELSE
      vecth(l) = g1DAr(tcp)%rovn(iptrs(i))
    ENDIF
  ENDDO
! Moderator Void Fraction:
  DO i = 1,numvols

l = l + 1
tcp = compptrs(i)
IF (is3d(i)) THEN
  vecth(l) = vsAr3(tcp)%alpn(iptrs(i),jptrs(i),kptrs(i))
ELSE
  vecth(l) = g1DAr(tcp)%alpn(iptrs(i))
ENDIF
ENDDO

! Moderator Boron Concentration:
DO i = 1,numvols
  l = l + 1
  tcp = compptrs(i)
  IF (is3d(i)) THEN
    vecth(l) = vsAr3(tcp)%conc(iptrs(i),jptrs(i),kptrs(i))
  ELSE
    vecth(l) = g1DAr(tcp)%concn(iptrs(i))
  ENDIF
  vecth(l) = vecth(l) * 1.0D+06
ENDDO

! Compute volume-average fuel temperature and copy
! to the vector of thermal-hydraulic data.
DO 20 i = 1,numhss
  l = l + 1
  tcp = hsptrs(i)
  tavg = 0.0D+00
  IF (hsaxial(i) .EQ. rodTab(tcp)%ncrz+1) THEN
    ztop = hsAr(tcp)%rdz(hsaxial(i))
    zbot = 0.5D+00 * (ztop + hsAr(tcp)%rdz(hsaxial(i)-1))
  ELSEIF (hsaxial(i) .EQ. 1) THEN
    zbot = hsAr(tcp)%rdz(1)
    ztop = 0.5D+00 * (zbot + hsAr(tcp)%rdz(2))
  ELSE
    zbot = 0.5D+00 * (hsAr(tcp)%rdz(hsaxial(i)) +
                     hsAr(tcp)%rdz(hsaxial(i)-1))
    ztop = 0.5D+00 * (hsAr(tcp)%rdz(hsaxial(i)) +
                     hsAr(tcp)%rdz(hsaxial(i)+1))
  ENDIF
  ri = 0.0D+00
  DO 10 j = 1,hsfsn(i)
    IF (j .EQ. hsfsn(i)) THEN
      ro = hsAr(tcp)%radrn(j,hsaxial(i),hsrods(i))
    ELSE
      ro = 0.5D+00 * (hsAr(tcp)%radrn(j,hsaxial(i),hsrods(i)) +
                      hsAr(tcp)%radrn(j+1,hsaxial(i),hsrods(i)))
    ENDIF
  IF (isslab(i)) THEN
    tavg = tavg + (hsAr(tcp)%rftn(j,hsaxial(i),hsrods(i))*
                    (ro - ri) * (ztop - zbot))
  ELSE
    tavg = tavg + (hsAr(tcp)%rftn(j,hsaxial(i),hsrods(i))*
                    (ro*ro - ri*ri) * (ztop - zbot))
  ENDIF
  ri = ro
10  CONTINUE
IF (isslab(i)) THEN
  tavg = tavg / (hsAr(tcp)%radrn(hsfsn(i),hsaxial(i),hsrods(i))
  &   * (ztop - zbot))
ELSE
  tavg = tavg / (hsAr(tcp)%radrn(hsfsn(i),hsaxial(i),hsrods(i))
  &   * hsAr(tcp)%radrn(hsfsn(i),hsaxial(i),hsrods(i))
  &   * (ztop - zbot))
ENDIF
vecth(l) = tavg
20 CONTINUE
! Fuel Centerline Temperature:
DO i = 1,numhss
  l = l + 1
  tcp = hsptrs(i)
  vecth(l) = hsAr(tcp)%rftn(1,hsaxial(i),hsrods(i))
ENDDO
! Fuel Surface Temperature:
DO i = 1,numhss
  l = l + 1
  tcp = hsptrs(i)
  vecth(l) = hsAr(tcp)%rftn(hsfsn(i),hsaxial(i),hsrods(i))
ENDDO

At this point, all data error-checking in this unit should be complete and the TDMR error logicals should accurately reflect the error status of the TDMR. These logicals are packed into the second and third words of the logical control buffer, and the time-dependent thermal-hydraulic control buffers are sent to the General Interface, as shown below:

lbufth(2) = errdatath
lbufth(3) = errpvmth
CALL TDMRSendBufth()

Once the control buffers have been sent and the vector of thermal-hydraulic and heat structure data has been constructed, this vector is packed into a PVM data structure, along with the vector dimension, and sent to the General Interface, as shown below:

istride = 1
CALL pvmfinitesend( PVMDEFAULT, info)
! Pack vector buffer dimension.
CALL pvmfpack( INTEGER4, nvecth, 1, istride, info)
! Pack unpermuted vector of thermal-hydraulic data.
CALL pvmfpack( REAL8, vecth, nvecth, istride, info)
CALL pvmfsend( tidgi, mtypeth, info)

II.E. General Interface to TRAC-M Mapping

This functional unit performs two basic tasks: (1) receive PARCS time-dependent control buffers, and (2) receive and store the permuted vector of neutronic data. The two subroutines utilized for these tasks, TDMRBufn2th() and TDMRMapn2th() respectively, are contained in module TDMRTIMECalc. The procedure of TDMRBufn2th() is similar to that used in
TDMRBufInit() for the receipt of the PARCS initial control buffers. The difference here is seen
in the data extracted from the buffers, as shown below. These data are described in the SRS for
the TDMR, and a description of the variables is given in the tables of Appendix A.

! Extract data from the PARCS control buffer.
errcalcn = lbufn(1)
errdatan = lbufn(2)
errpvnn = lbufn(3)
errdatag = lbufn(4)
errpvmg = lbufn(5)
pdmrdone = lbufn(6)
IF (ssinit .AND. cbccalc) THEN
            cbcratio = r8bufn(1)
ENDIF

The first step of Task (2) is to receive the data structure sent from the General Interface,
which is composed of the permuted vector of neutronic data and the vector dimension. Once
received, the vector dimension is unpacked first and used to allocate space for the permuted vec-
tor, which is allocated only once at the beginning of the time-dependent calculation. The following
lines of code demonstrate this procedure:

CALL pvmftrecv( tidgi, mtypegi, timeout, 0, info)
istride = 1
! Unpack vector buffer dimension and allocate memory for the vector.
CALL pvmfunpack( INTEGER4, nvecnp, 1, istride, info)
IF (.NOT.ALLOCATED(vecnp))  ALLOCATE(vecnp(nvecnp))
! Unpack permuted vector of Neutronic data.
CALL pvmfunpack( REAL8, vecnp, nvecnp, istride, info)
IF (.NOT.errdatath) THEN
     DO ii = 1,numvols
            IF (.NOT.is3d(ii)) THEN
                pipeTab(compptrs(ii))%cpow = 0.0D+00
            ELSE
                vsAr3(compptrs(ii))%q3drl(iptrs(ii),jptrs(ii),kptrs(ii)) = 0.0D+00
                vsAr3(compptrs(ii))%q3drv(iptrs(ii),jptrs(ii),kptrs(ii)) = 0.0D+00
            ENDIF
     ENDDO
     DO ii = 1,numvols
            IF (.NOT.is3d(ii)) THEN
                pipeTab(compptrs(ii))%cpow = pipeTab(compptrs(ii))%cpow +
                & vecnp(ii)
            ELSE
                vsAr3(compptrs(ii))%q3drl(iptrs(ii),jptrs(ii),kptrs(ii)) = 
                & vsAr3(compptrs(ii))%q3drl(iptrs(ii),jptrs(ii),kptrs(ii)) +
                & vecnp(ii) * (1.0D+00 -
                & vsAr3(compptrs(ii))%alpn(iptrs(ii),jptrs(ii),kptrs(ii)) /
                & (1.0D+00 - alpcut)
                vsAr3(compptrs(ii))%q3drv(iptrs(ii),jptrs(ii),kptrs(ii)) = 
                & vsAr3(compptrs(ii))%q3drv(iptrs(ii),jptrs(ii),kptrs(ii)) +
                & vecnp(ii) *
                & vsAr3(compptrs(ii))%alpn(iptrs(ii),jptrs(ii),kptrs(ii)) /
                & (1.0D+00 - alpcut)
ENDIF
ENDDO
IF (naxn .GT. 0) THEN
  corePowerG%powModerTot = 0.0D+00
  corePowerG%powModerVess = 0.0D+00
  corePowerG%powModerChans = 0.0D+00
  corePowerG%rpower = 0.0D+00
  corePowerG%rpowrn = 0.0D+00
  corePowerG%rpowr = 0.0D+00
  DO ii = 1,numvols
    IF (ischanth(ii)) THEN
      corePowerG%powModerChans = corePowerG%powModerChans +
      & vecnp(ii)
    ELSE IF (is3d(ii)) THEN
      corePowerG%powModerVess = corePowerG%powModerVess +
      & vecnp(ii)
    ENDIF
    corePowerG%powModerTot = corePowerG%powModerTot +
    & vecnp(ii)
  ENDDO
  DO ii = 1,(numvols+numhss)
    corePowerG%rpower = corePowerG%rpower + vecnp(ii)
    corePowerG%rpowrn = corePowerG%rpowrn + vecnp(ii)
    corePowerG%rpowr = corePowerG%rpowr + vecnp(ii)
  ENDDO
ENDIF
DO ii = 1,numhss
  rodTab(hsptrs(ii))%rpowrn = 0.0D+00
  rodTab(hsptrs(ii))%rpower = 0.0D+00
ENDDO
DO ii = 1,numhss
  rodTab(hsptrs(ii))%rpowrn = rodTab(hsptrs(ii))%rpowrn +
  & vecnp(ii + numvols)
  rodTab(hsptrs(ii))%rpower = rodTab(hsptrs(ii))%rpower +
  & vecnp(ii + numvols)
ENDDO
ENDIF
IF (ssinit .AND. cbccalc .AND. .NOT.errdatath) CALL TDMR_Adj_Boron()

The next step is to map these data to the appropriate memory locations used by TRAC-M. If
the current calculation is a steady-state initialization calculation, the subroutine responsible for
adjusting the boron concentration, TDMRAdjBoron, must be called. The mapping performed in
this subroutine is shown below. Two types of data are sent from PARCS (in order): power depos-
itied into the coolant and power deposited into the fuel. The method for mapping the power depos-
itied into the coolant is handled within modules Pipe, VessTF3DS, and Gen1DCrunch, which
contain the subroutines necessary for calculating direct power deposition within the coolant. The
mapping of the power deposited into the fuel is handled within the TRAC-M module Rod-
Crunch, which contains all the subroutines for performing heat conduction calculations. In order
to accomodate the mapping of PARCS neutronics data to the thermal-hydraulic nodalization, sub-
routine pipe1 in module Pipe, subroutine tf3ds in module VessTF3DS, subroutine tf1ds in
module Gen1DCrunch, and subroutine rodht in module RodCrunch needed to be modified.
The modifications made to these subroutines will be described below.
II.E.1. Critical Boron Concentration Adjustments

If a critical boron concentration calculation is to be performed by the coupled code, the boron present in the TRAC-M model must be adjusted by the ratio sent by PARCS in the time-dependent neutronic control buffer. Subroutine TDMRAdjBoron in module TDMRTimcCalc is responsible for this task. The algorithm shown below loops over all the components in the TRAC-M system, as well as all the cells making up each component, and multiplies the boron concentration by the ratio sent from PARCS for each cell. It should be noted that this algorithm preserves zero boron concentrations in the TRAC-M system, which is important for secondary-side components. Time-dependent boron concentration tables must also be adjusted to preserve continuity during a transient calculation.

DO ll = 1,ncomp
   tcp = compIndices(ll)
   Select_Comp: SELECT CASE (INT(genTab(tcp)%type))
   CASE(INT(pipeh))
      IF (pipeTab(tcp)%iconc .NE. 0) THEN
         DO ii = 1,pipeTab(tcp)%ncells
            g1DAr(tcp)%conc(ii) = g1DAr(tcp)%conc(ii) * cbcratio
         ENDDO
      ENDIF
   CASE(INT(vsslh))
      IF (vessTab(tcp)%iconc .NE. 0) THEN
         DO ii = 1,vessTab(tcp)%ncells
            DO jj = 1,vessTab(tcp)%ncells
               DO kk = 1,vessTab(tcp)%ncells
                  vsAr3(tcp)%conc(ii,jj,kk) = vsAr3(tcp)%conc(ii,jj,kk) * cbcratio
               ENDDO
            ENDDO
         ENDDO
      ENDIF
   CASE(INT(pumph))
      IF (pumpTab(tcp)%iconc .NE. 0) THEN
         DO ii = 1,pumpTab(tcp)%ncells
            g1DAr(tcp)%conc(ii) = g1DAr(tcp)%conc(ii) * cbcratio
         ENDDO
      ENDIF
   CASE(INT(teeh))
      IF (((teeTab(tcp)%iconc1 .NE. 0) .OR. (teeTab(tcp)%iconc2 .NE. 0)) THEN
         DO ii = 1,teeTab(tcp)%ncells
            g1DAr(tcp)%conc(ii) = g1DAr(tcp)%conc(ii) * cbcratio
         ENDDO
      ENDIF
   ENDCASE
ENDDO
ENDIF
CASE(INT(valveh))
  IF (valveTab(tcp)%iconc .NE. 0) THEN
    DO ii = 1,valveTab(tcp)%ncells
      g1DAr(tcp)%concn(ii) = g1DAr(tcp)%concn(ii) * cbcratio
      g1DAr(tcp)%conc(ii) = g1DAr(tcp)%concn(ii)
    ENDDO
  ENDIF
CASE(INT(prizrh))
  IF (prizeTab(tcp)%iconc .NE. 0) THEN
    DO ii = 1,prizeTab(tcp)%ncells
      g1DAr(tcp)%concn(ii) = g1DAr(tcp)%concn(ii) * cbcratio
      g1DAr(tcp)%conc(ii) = g1DAr(tcp)%concn(ii)
    ENDDO
  ENDIF
CASE(INT(sepdh))
  IF ((teeTab(tcp)%iconc1 .NE. 0) .OR.
    &   (teeTab(tcp)%iconc2 .NE. 0)) THEN
    DO ii = 1,teeTab(tcp)%ncells
      g1DAr(tcp)%concn(ii) = g1DAr(tcp)%concn(ii) * cbcratio
      g1DAr(tcp)%conc(ii) = g1DAr(tcp)%concn(ii)
    ENDDO
  ENDIF
CASE(INT(breakh))
  IF (breakTab(tcp)%ibty .EQ. 5) THEN
    CALL scltbl(breakAr(tcp)%contb,'conctbab',
    &      IABS(breakTab(tcp)%nbtb,1,cbcratio,breakTab(tcp)%ibsv)
    breakTab(tcp)%conoff = breakTab(tcp)%conoff * cbcratio
  ENDIF
  g1DAr(tcp)%concn(1) = g1DAr(tcp)%concn(1) * cbcratio
  g1DAr(tcp)%conc(1) = g1DAr(tcp)%concn(1)
CASE(INT(fillh))
  IF (((fillTab(tcp)%ifty .EQ. 6) .OR.
    & (fillTab(tcp)%ifty .EQ. 9)) THEN
    CALL scltbl(fillTab(tcp)%contb,'conctbab',
    &      IABS(fillTab(tcp)%nftb,1,cbcratio,fillTab(tcp)%ifsv)
    fillTab(tcp)%conoff = fillTab(tcp)%conoff * cbcratio
  ENDIF
  g1DAr(tcp)%concn(1) = g1DAr(tcp)%concn(1) * cbcratio
  g1DAr(tcp)%conc(1) = g1DAr(tcp)%concn(1)
CASE(INT(plenh))
  IF (plenTab(tcp)%iconc .NE. 0) THEN
    g1DAr(tcp)%concn(1) = g1DAr(tcp)%concn(1) * cbcratio
    g1DAr(tcp)%conc(1) = g1DAr(tcp)%concn(1)
  ENDIF
END SELECT Select_Comp
ENDDO
It should be noted that TURBINE components are omitted from this listing because of their absence from the TRAC-M source, i.e., all TURBINE component coding is commented out. Subroutine TDMRAdjBoron() is called during every call to TDMR(3) if the current calculation is a steady-state initialization calculation and a critical boron concentration calculation is being performed by PARCS.

II.E.2. Power Deposition Within a PIPE/CHAN

The coding modifications within TRAC-M necessary to allow power deposition within the cells of a PIPE/CHAN component are presented in this section, along with subroutines TDMRPipeCPow and TDMRPipeQ in module TDMRTimeCalc, which actually perform the energy deposition calculation. The original coding in TRAC-M from subroutine pipe1 in module Pipe is shown and discussed below:

```fortran
IF (.NOT.((im100.EQ.-100) .OR. (pipeTab(cco)%ipow.EQ.0))) THEN
   pipeTab(cco)%eninp=pipeTab(cco)%eninp+odelt*pipeTab(cco)%cpow
   xd(1)=pipeTab(cco)%cpow
   CALL evfxxx(xd, ...)
   pipeTab(cco)%cpow=xd(1)
ENDIF
```

The code fragment above evaluates the total power deposited in the coolant within a pipe. This quantity is stored in the variable cpow, which is calculated (interpolated if necessary) from the user input table using subroutine evfxxx. Subroutine pipe1 is called once for each PIPE/CHAN component making up the TRAC-M model, so slight modifications to this fragment allow direct PIPE/CHAN coolant energy deposition to be implemented easily. First, a logical variable, coupled, is set to .FALSE. at the beginning of this subroutine to signify whether the current PIPE component is coupled with the kinetics calculation. The modified code, then, is as follows:

```fortran
IF (.NOT.((im100.EQ.-100) .OR. (pipeTab(cco)%ipow.EQ.0))) THEN
   IF (itdmr.EQ.1) THEN
      CALL TDMRPipeCPow(cco, coupled)
   ENDIF
   IF (coupled) GOTO 100
   xd(1)=pipeTab(cco)%cpow
   CALL evfxxx(xd, ...)
   pipeTab(cco)%cpow=xd(1)
100 CONTINUE
ENDIF
```

This modification allows entry into the TDMRPipeCPow subroutine for all PIPE/CHAN components, but only those which are coupled to the kinetics calculation follow an altered calculational flow. Furthermore, if TRAC-M is run in stand-alone mode (with itdmr=0), the calculational flow is completely unaffected by this change. The algorithm implemented in subroutine TDMRPipeCPow(tcp, cpld) is shown below:
DO ll = 1,numvols
  IF (compptrs(ll) .EQ. tcp)  GOTO 50
ENDDO
GOTO 60
50 CONTINUE
DO mm = ll,numvols
  pipeTab(tcp)%eninp = pipeTab(tcp)%eninp +
    & odelt * pipeTab(tcp)%cpow
ENDDO
  cpld = .TRUE.
60 CONTINUE

The most notable feature of this algorithm is that the original functionality of the TRAC-M calculation flow is retained. If the current component is not coupled to the kinetics calculation, the subroutine returns with the logical variable cpld (coupled in subroutine pipe1) still set to .FALSE., which indicates that subroutine pipe1 should evaluate the power table itself.

TRAC-M assumes that energy deposition within the coolant is constant within a PIPE/CHAN component, which is inconsistent with the desired functionality of the coupled TRAC-M/PARCS code. The ratio of the energy deposited within a particular cell of a PIPE/CHAN to the total energy deposited the entire PIPE/CHAN is equal to the ratio of length of that particular cell to the total length of the PIPE/CHAN. Since the coupling of each cell within the PIPE/CHAN is explicitly coupled to the kinetics calculation through the mapping information input file, MAPTAB, the need to specify the power being deposited into each cell within the PIPE/CHAN exists. Subroutine TDMRPipeQ(tcp, cell, alp, alpcut, qlp, qvp, deltat, cpld) was created for this purpose and is shown below:

DO ll = 1,numvols
  IF ((compptrs(ll) .EQ. tcp) .AND.
& (iptrs(ll) .EQ. cell))  GOTO 70
ENDIF
GOTO 80
70 CONTINUE
  cpld = .TRUE.
  IF (alp .LT. alpcut) THEN
    qlp = vecnp(ll) * deltat
    qvp = 0.0D+00
  ELSE
    qlp = ((1.0D+00 - alp)/(1.0D+00 - alpcut) * vecnp(ll) * deltat
    qvp = (vecnp(ll) * deltat) - qlp
  ENDIF
80 CONTINUE
! We need to check whether this cell just isn’t mapped to kinetics.
! In this case, it is treated as a coupled cell, and the relevant
! data are set to 0.
  IF (.NOT.cpld) THEN
    DO ll = 1,numvols
      IF (compptrs(ll) .EQ. tcp) THEN
        cpld = .TRUE.
        qlp = 0.0D+00
        qvp = 0.0D+00
    ENDIF
  ENDIF
The coding above is only for cells coupled to the kinetics calculation (or those cells in a component partly mapped to kinetics) and is meant to supersede the coding in subroutine pipe2 in module Pipe shown below:

\[ q_{tp} = \text{pipeTab}(ccp)\%cpow \times \text{delt} / \text{pipeTab}(cco)\%plent \]

Where \( \text{plent} \) is the total length of the pipe and \( q_{tp} \) is used in subroutine \text{td1ds} in module \text{Gen1DCrunch} shown below:

\[
\text{IF (qtp.NE.0.d0) THEN}
\text{IF (alp(j).LT.alpcut) THEN}
\quad qlp=qtp*\text{dx}(j)
\quad qvp=0.d0
\text{ELSE}
\quad qlp=(1.0d0-alp(j))/(1.0d0-alpcut)*qtp*\text{dx}(j)
\quad qvp=qtp*\text{dx}(j)-qlp
\text{ENDIF}
\quad q1=q1+qlp
\quad qv=qv+qvp
\text{ENDIF}
\]

Subroutine \text{tf1ds} was modified to allow these changes in calculation for coupled cells in the manner shown below:

\[
\text{IF (qtp.NE.0.d0) THEN}
\text{IF (itdmr.EQ.1) THEN}
\quad \text{CALL TDMRPipeQ(cco,j,alp(j),alpcut,qlp,qvp,delt,coupled)}
\text{ENDIF}
\text{IF (coupled) GOTO 100}
\text{IF (alp(j).LT.alpcut) THEN}
\quad qlp=qtp*\text{dx}(j)
\quad qvp=0.d0
\text{ELSE}
\quad qlp=(1.0d0-alp(j))/(1.0d0-alpcut)*qtp*\text{dx}(j)
\quad qvp=qtp*\text{dx}(j)-qlp
\text{ENDIF}
\quad q1=q1+qlp
\quad qv=qv+qvp
\text{ENDIF}
\]

\text{100 CONTINUE}
\quad q1=q1+qlp
\quad qv=qv+qvp
\text{ENDIF}

Thus, the original calculational flow is preserved for components which are not coupled with the kinetics calculation as specified in the mapping input file, MAPTAB.

II.E.3. Power Deposition Within a VESSEL
The coding modifications necessary to allow direct energy deposition into the coolant of a VESSEL component are presented here. All the necessary coding was actually performed already after the initial delivery date of the TRAC-M/PARCS code. Therefore, adding this functionality was as simple as placing the data from PARCS in the proper TRAC-M memory location. Subroutine tf3ds in module VessTF3DS performs the calculations for energy deposition in the coolant. The relevant coding is shown below:

```fortran
IF (itdmr.GT.1) THEN
  q1(k)=q1(k)+vsAr3(cco)%q3dr1(i,j,k)*delt
  qv(k)=qv(k)+vsAr3(cco)%q3drv(i,j,k)*delt
  vsAr3(cco)%q3direct(i,j,k)=0.0D+00
ENDIF
```

The power to be deposited in the VESSEL coolant is stored in `vsAr3(cco)%q3dr1(:,,:,:)` and `vsAr3(cco)%q3drv(:,,:,:)`, and was shown in Section II.E.

### II.E.4. Power Deposition Within a Heat Structure

Two subroutines needed to be modified in order to accurately map the heat generation rates as calculated by PARCS to the TRAC-M components. Subroutine `core1` in module `RodTask` needed to be modified so that the point kinetics calculation can be skipped over if a heat structure is coupled to the kinetics calculation. The partial listing of the original coding in this subroutine is listed below:

```fortran
IF (rodTab(cco)%nopwr.NE.0.OR.ischan) GOTO 370
! determine the reactor core power
  power=rodTab(cco)%rpower
  ...
  <point kinetics calculation, power table lookup, power shape evaluation>
  ...
370 CONTINUE
RETURN
```

The FORTRAN statements between the test for absence of power generation within a heat structure (`nopwr.EQ.0`) and statement 370 may also be skipped if a heat structure is coupled to the kinetics calculation. Therefore, the coding in `core1` was changed to the following:

```fortran
IF (rodTab(cco)%nopwr.NE.0.OR.ischan.OR.rodTab(cco)%irpwty.EQ.8)
&    GOTO 370
! determine the reactor core power
  power=rodTab(cco)%rpower
  ...
  <point kinetics calculation, power table lookup, power shape evaluation>
  ...
370 CONTINUE
RETURN
```
It should be noted that the variable for defining reactivity and power type for coupled heat structures (irpwty) is set to 8 when the heat structure composite ID numbers are decoded in the error-checking module. Thus, the original functionality is preserved throughout this modification.

Subroutine rodht in module RodCrunch calculates the volumetric heat generation rate at each radial node on the axial levels of the heat structures. Ordinarily, this subroutine uses radial and axial power density shapes based on user input to distribute power generated within a heat structure amongst all its nodes. When a heat structure is coupled to a kinetics calculation, it is desirable to override the axial power shape, as the axial power shape is inherent in the 3-D kinetics data. The radial power distribution of a heat structure, however, still must be used to distribute the power generated within an axial level. Due to this change in the calculational flow, modifications were made to subroutine rodht.

The calculation of the volumetric heat generation rate within rodht is shown below:

\[ qppp(i,jbd) = powr \cdot rpowf(i) + qmetw \]

The variables involved in this calculation of the volumetric heat generation rate (powr and rpowf(i)) have the axial and radial power shapes already factored into them, thus disallowing this implementation the advantage of using spatial kinetics data. This calculation was modified in rodht to the following to incorporate spatial kinetics data into this subroutine:

\[ \text{CALL TDMRHtstrQppp(tcp, ncr, j, i, z, r, rpwf, rnr, nr, nz, matrd, qppp(i, jbd))} \]
\[ qppp(i, jbd) = qppp(i, jbd) + qmetw \]

Subroutine TDMRHtstrQppp(tcp, ncr, axial, radial, z, r, rpwf, actual-rods, rnr, nz, matrid, volq) is part of module TDMRTimeCalc, which was created to calculate the volumetric heat generation rate at each radial node within the axial levels of a heat structure. This coding for this subroutine is listed and discussed below:

DO mm = 1,numhss
  IF ((hsptrs(mm) .EQ. cco) .AND. (hsrods(mm) .EQ. rodnum) .AND. (hsaxial(mm) .EQ. axial)) THEN
    IF (.NOT.isrefl(mm)) THEN
      cpld = .TRUE.
      GOTO 90
    ENDIF
  ENDIF
ENDDO
GOTO 100

The first task is to find the location in the permuted vector of neutronic data which relates to the current heat structure, average rod, and axial level. The value of the current component pointer, cco, is obtained by including module GlobalDat in a USE statement. The average ROD/SLAB number, rodnum, can be tracked by incrementing this variable only if the current node is on the outer surface of the highest axial level within a heat structure. By matching these three identification parameters, the correct power for this rod may be found in the vector of neutronic data, vecnp. It should be noted that the ordering scheme used in the communication of
data within the coupled code for ensuring data consistency is compatible with the order used by TRAC-M for calculating the conduction solution.

90 CONTINUE
   IF (radial .EQ. 1) THEN
      ALLOCATE( nodeareas(nrn))
      ALLOCATE( nodevols(nrn))
      ALLOCATE( wtdareas(nrn))
      ALLOCATE( mycqs(nrn))
      ALLOCATE( temprpowf(nrn))
   ENDIF

   Space is then allocated for some temporary calculational variables if this heat conduction node is along the centerline. Because TRAC-M calculates heat generation rate from the centerline outward, this allocation is safe.

   IF (axial .EQ. 1) THEN
      zbot = z(axial)
      ztop = 0.5D+00 * (z(axial+1) + z(axial))
   ELSE IF (axial .EQ. nzn) THEN
      zbot = 0.5D+00 * (z(axial-1) + z(axial))
      ztop = z(axial)
   ELSE
      zbot = 0.5D+00 * (z(axial-1) + z(axial))
      ztop = 0.5D+00 * (z(axial+1) + z(axial))
   ENDIF

   The next task is to calculate the ‘height’ of the current heat conduction axial level. Each node represents a certain finite volume, and calculating the height is an intermediate step toward calculating the “volume” of each node. If this radial node is along the centerline, many temporary calculational parameters need to be computed in order to calculate the volumetric heat generation rate at a specific radial conduction node.

   IF (radial .EQ. 1) THEN
      tot = 0.0D+00
      DO ll = 1,nrn
         tot = tot + rpwf(ll)
      ENDDO
      DO ll = 1,nrn
         temprpowf(ll) = rpwf(ll) / tot
      ENDDO
  ENDIF

   This portion of code simply normalizes the radial power distribution (passed to and stored by this subroutine as rpwf) so that the summation is equal to unity. The radial power distribution is taken from user-input for each heat structure.

   DO ll = 1,nrn
      IF (ll .EQ. 1) THEN
         ir = 0.0D+00
         ro = 0.5D+00 * (r(ll+1,axial) + r(ll,axial))
      mycqs(ll) = 1.0D+00
   ENDIF
ELSE IF (ll .EQ. nrn) THEN
  ir = 0.5D+00 * (r(ll-1,axial) + r(ll,axial))
  ro = r(ll,axial)
  mycqs(ll) = 1.0D+00
ELSE
  ir = 0.5D+00 * (r(ll-1,axial) + r(ll,axial))
  ro = 0.5D+00 * (r(ll+1,axial) + r(ll,axial))

Next, the inner and outer radii \( ir \) and \( ro \), respectively, of the current heat conduction “volume” is calculated, as shown above. The next task relates to identifying the fraction of the conduction volume which is actually solid material. This calculation is necessary only in the presence of a gas gap definition (the gas gap is denoted by an entry equal to 3 in the material identification array, \textit{matrid}).

\[
\text{IF (INT(matrid(ll)) .EQ. 3) THEN}
\text{  IF (isslab(mm)) THEN}
\text{    mycqs(ll) = (r(ll,axial) - ir) / (ro - ir)}
\text{  ELSE}
\text{    mycqs(ll) = (r(ll,axial) * r(ll,axial) - ir * ir) / (ro * ro - ir * ir)}
\text{  ENDIF}
\text{ELSE IF (INT(matrid(ll-1)) .EQ. 3) THEN}
\text{  IF (ll .GT. hsfsn(mm)) THEN}
\text{    mycqs(ll) = 1.0D+00}
\text{  ELSE}
\text{    IF (isslab(mm)) THEN}
\text{      mycqs(ll) = (ro - r(ll,axial)) / (ro - ir)}
\text{    ELSE}
\text{      mycqs(ll) = (ro * ro - r(ll,axial) * r(ll,axial)) / (ro * ro - ir * ir)}
\text{    ENDIF}
\text{  ENDIF}
\text{ENDIF}
\text{ELSE}
\text{  mycqs(ll) = 1.0D+00}
\text{ENDIF}
\text{ENDIF}
\text{ENDIF}
\text{ENDIF}
\text{ENDIF}
\text{ENDIF}
\text{ENDIF}
\text{ENDIF}
\text{ENDIF}
\text{ENDIF}

The fraction \( cq \) is simply the fraction of the current conduction volume which is solid material. Figure 8 shows a typical nodalization (not to scale) of a ROD component. The fifth and sixth radial nodes represent conduction volumes which overlap the gas gap. However, no consideration of this fact is made in the original TRAC-M calculation of the conduction volume. The parameter \( cq \) defines the fraction of conduction volumes 5 and 6 which is occupied by fuel material. This parameter is important because the volume of the gas gap should not be considered in the calculation of the conduction volume for a particular node. The coding above demonstrates the algorithm needed to accurately calculate this parameter.
Calculation of the cross-sectional areas of each conduction volume is the next task so that area-weighting of the normalized radial power distribution may be performed.

```
IF (isslab(mm)) THEN
    nodeareas(ll) = rodTab(hsptrs(mm))%width * (ro - ir)
ELSE
    nodeareas(ll) = pi * (ro * ro - ir * ir)
ENDIF
ENDDIF
ENDDO
```

Figure 8: Demonstration of parameter CQ

```
DO ll = 1,nrn
    wtdareas(ll) = nodeareas(ll) * mycqs(ll)
ENDDO
```
\[ \text{tot} = 0.0D+00 \]
\[ \text{DO ll = 1,nrn} \]
Furthermore, each node area should be weighted by the parameter $c_q$ then normalized, yielding an “effective area,” i.e., the fraction each node represents of the total heat structure fuel cross-section.

Furthermore, each node area should be weighted by the parameter $c_q$ then normalized, yielding an “effective area,” i.e., the fraction each node represents of the total heat structure fuel cross-section.

The normalized radial power distribution should be effective-area-weighted and normalized in order to determine the fraction of the total heat generated in this axial level at the radial node of interest.

The CHAN component causes some difficulty with the $n_{chans}$ parameter, which represents the number of actual channels in the BWR core being modelled by the CHAN component in question. This translates directly to the actual number of fuel pins that are represented, so special coding needs to account for this fact, as shown below:

Finally, the volume of each node needs to be calculated in order to determine the volumetric heat generation rate at each radial node.

```plaintext
DO ll = 1,nrn
    nodevols(ll) = nodeareas(ll) * (ztop - zbot)
ENDDO
ENDIF

IF (vecnp(mm+numvols) .EQ. 0.0D+00) THEN
    volq = 0.0D+00
ELSE
    volq = (vecnp(mm+numvols) * temprpowf(radial)) / &
            (nodevols(radial) * mycqs(radial) * actualnumrods)
ENDIF
```
The volumetric heat generation rate at the radial node of interest (node number radial) is determined from the total power deposited into this axial level of the heat structure (given by $\text{vecnp}(\text{mm+numvols})$) multiplied by the fraction of the total power which should appear at this radial node (given by $\text{temprpowf(radial)}$) divided by the volume of the conduction node ($\text{nodevols(radial)}$) divided by the area fraction of solid material in this conduction node ($\text{mycqs(radial)}$) divided by the actual number of rods this average rod represents (actual-rods).

IF (axial .EQ. nzn .AND. radial .EQ. nrn) THEN
    IF (rodnum .EQ. rodTab(cco)%ncrx) THEN
        rodnum = 1
    ELSE
        rodnum = rodnum + 1
    ENDIF
ENDIF
IF (radial .EQ. nrn) THEN
    DEALLOCATE( nodeareas)
    DEALLOCATE( nodevols)
    DEALLOCATE( wtdareas)
    DEALLOCATE( mycqs)
    DEALLOCATE( temprpowf)
ENDIF

The final step in this calculation is updating the value of rodnum correctly and deallocating memory no longer needed.

II.F. Error-Checking

The error-checking module, TDMRErrorCheck, contains three subroutines, TDMRDataErrchk(), TDMRPVMErrchk(), and TDMRProcErrchk(), which are called from each of the previously described functional units. The error-checking operations performed by TDMRDataErrchk() are specific to each unit, and the call to this subroutine includes one argument, errcode, which is an integer relating to the specific error check to be performed (shown in parenthesis below). This subroutine also decodes the data sent in the initial neutronic control buffer, specifically, the composite ID numbers formed by the PDMR. Pointers to all the important data are found and stored in TDMR variables for use throughout the TDMR functionality. If an error is detected in this subroutine, the logical errdatath (see Table 6 of Appendix A) is set to .TRUE., and the appropriate error message is displayed. In addition, the TDMR then communicates a “data” error, which indicates that the calculation should be terminated, to the General Interface process, and consequently the PARCS process, using the space available in the thermal-hydraulic logical buffer, lbufth.

Initialization:

(1) [warning] No trips are defined in the TRAC-M input deck:
result from: no trips exist in the TRAC-M input deck but a trip is specified in the the mapping input file, MAPTAB
(2) **[warning]** The trip card defined in the mapping input file, MAPTAB, does not exist in the TRAC-M input deck:  
result from: the trip ID number specified in the mapping information does not exist in the TRAC-M model

(3) **[warning]** A trip ID was received which is outside the bounds of a legal trip ID  
result from: itrip .LT. 0 or itrip .GE. 10000

(4) **[fatal error]** A component number defined in %TABLE1 does not exist in the TRAC-M input deck:  
result from: a component number specified in the thermal-hydraulic mapping table %TABLE1 does not exist in the TRAC-M model

(5) **[fatal error]** A component number specified is not a thermal-hydraulic component:  
result from: a component number specified in the thermal-hydraulic mapping table %TABLE1 is neither a VESSEL nor a PIPE

(6) **[fatal error]** A horizontal plane cell specified does not exist in the TRAC-M input:  
result from: a horizontal plane cell specified in the thermal-hydraulic mapping table %TABLE1 is greater than the number of actual horizontal plane cells in the VESSEL model or less than 0

(7) **[fatal error]** An axial level specified does not exist in the TRAC-M input:  
result from: an axial level specified in the thermal-hydraulic mapping table %TABLE1 is greater than the actual number of axial levels in the VESSEL model or less than 0

(8) **[warning]** The option for power deposited directly into the coolant was not set.  
result from: a pipe specified in the thermal-hydraulic mapping table %TABLE1 does not have the option for direct coolant heating set

(9) **[fatal error]** A PIPE cells specified does not exist in the TRAC-M input:  
result from: a PIPE cell specified in the thermal-hydraulic mapping table %TABLE1 is greater than the number of actual cells making up the PIPE model or less than 0

(10) **[fatal error]** Every cell of a PIPE component was not specified in %TABLE1:  
result from: a PIPE component with entries in %TABLE1 in MAPTAB does not have entries for every cell

(11) **[fatal error]** A component number defined in %TABLE2 does not exist in the TRAC-M input deck:  
result from: a component number specified in the heat structure mapping table %TABLE2 does not exist in the TRAC-M model

(12) **[fatal error]** A component number specified is not a heat structure component:
result from: a component number specified in the heat structure mapping table
%TABLE2 is neither a ROD, SLAB, or CHAN

(13) [fatal error] A heat structure has no heat source defined in the TRAC-M input deck:
result from: the input option nopowr is set to 1 for a heat structure component specified
in the heat structure mapping table %TABLE2

(14) [warning] A heat structure has the option for axial mesh refinement activated:
result from: the input option irftr is not set to 0, which allows the heat structure
nodalization to dynamically change according to a trip

(15) [fatal error] Only one radial node exists for the heat structure:
result from: the input option nodes is set to 1, which disallows the definition of a fuel
surface node for a heat structure specified in the heat structure mapping
table %TABLE2

(16) [warning] No fuel surface detected for specified heat structure:
result from: a heat structure specified in the heat structure mapping table %TABLE2
has a radial power distribution with all non-zero values

(17) [fatal error] An average RODs/SLABs does not exist in the TRAC-M input:
result from: an average ROD/SLAB specified in in the heat structure mapping table
%TABLE2 is greater than the actual number of average RODs/SLABs
making up the heat structure or is less than 0

(18) [fatal error] An axial level specified does not exist in the TRAC-M input:
result from: an axial level specified in the heat structure mapping table %TABLE2 is
greater than the actual number of axial levels in the heat structure or is
less than 0

(19) [fatal error] The average RODs/SLABs of a heat structure were not completely defined
in %TABLE2:
result from: a heat structure with multiple average ROD/SLAB elements does not
have all ROD/SLAB elements mapped to the kinetics calculation

(20) [fatal error] The axial levels of a heat structure were not completely defined in
%TABLE2:
result from: every axial level of a heat structure was not specified in the heat structure
mapping table %TABLE2

(21) [warning] A boron criticality search is being performed but solute tracking is not on:
result from: the initial neutronic control buffer specified that a critical boron
concentration calculation is to be performed, but the option for solute
tracking is not activated in the TRAC-M model

(22) [warning] A boron criticality search is being performed but initial boron
concentration is 0:
result from: all the thermal-hydraulic cells defined in %TABLE1 of MAPTAB have an initial boron concentration of 0.0D+00

Thermal-Hydraulic to Neutronic Mapping:

(23) [fatal error]  Inconsistency between matrix and vector dimensions:
result from: nvecth ≠ MAXVAL(matcolth)

(24) [fatal error]  Property data extracted from permuted vector outside specified bounds:
result from:
(24) 0.0 K > Mod. Temp. > 10000.0 K
(24) 0.0 kg/m^3 > Mod. Liquid Density > 5000.0 kg/m^3
(24) 0.0 kg/m^3 > Mod. Vapor Density > 5000.0 kg/m^3
(24) 0.0 > Void Fraction > 1.0
(24) 0.0 ppm > Boron Concentration > 1.0E+06 ppm
(24) 0.0 K > Fuel Temp. > 10000.0 K

Neutronic to Thermal-Hydraulic Mapping:

(25) [fatal error]  Inconsistency between matrix and vector dimensions:
result from: nvecnp ≠ MAXVAL(matrown)
(25) nvecnp ≠ SIZE(vecnp)  (This check is required because the memory for vecnp is allocated only once--at the beginning of the time-dependent calculation. Thus, if the dimension, nvecnp, is not consistent with the size of the previously allocated vector, the unpacking of the vector will be in error.

(26) [fatal error]  Neutronic powers in permuted vector are outside specified bounds:
result from: 0.0 watts >   Power >   1.0E+15 watts

It should be noted that the error bounds shown in “data” errors 17 and 19 are not based on physical property constraints; it is expected that TRAC-M will fail on extreme data such as these. Rather, these bounds are simply utilized to trap errors which may have occurred either in the General Interface mapping procedure or during communication.

TDMRPMERRchk() detects errors in the PVM calls, and includes three arguments, errcode, iunit, and istat. errcode refers to the PVM function which returned an error; iunit refers to the subroutine calling TDMRPMERRchk(); and istat is the integer PVM status code for errcode and iunit. If an error is detected in this subroutine, the logical errpvmth (see Table 6 of Appendix A) is set to .TRUE., and the appropriate error message is displayed. The TDMR then attempts to communicate a “PVM” error to the General Interface process, and consequently the PARCS process, using the space available in the thermal-hydraulic logical
buffer, lbufth. It should be noted that an error resulting from a PVM call may not allow for safe termination of all processes.

<table>
<thead>
<tr>
<th>errcode:</th>
<th>associated with:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>pvmfinitsend</td>
</tr>
<tr>
<td>2</td>
<td>pvmfpack</td>
</tr>
<tr>
<td>3</td>
<td>pvmfsend</td>
</tr>
<tr>
<td>4</td>
<td>pvmftrecv</td>
</tr>
<tr>
<td>5</td>
<td>pvmfunpack</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iunit:</th>
<th>associated with:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PVM operations required in TDMR_Obtain_IDS()</td>
</tr>
<tr>
<td>2</td>
<td>PVM operations required in TDMR_Send_Bufth()</td>
</tr>
<tr>
<td>3</td>
<td>PVM operations required in TDMR_Recv_Bufn()</td>
</tr>
<tr>
<td>4</td>
<td>PVM operations required in TDMR_Send_Mat()</td>
</tr>
<tr>
<td>5</td>
<td>PVM operations required in TDMRMapn2th() to receive the permuted neutronic vector</td>
</tr>
<tr>
<td>6</td>
<td>PVM operations required in TDMRMapth2n() to send the unpermuted thermal-hydraulic vector</td>
</tr>
</tbody>
</table>

Finally, TDMRProcErrchk() performs checks on the value of the error logicals sent from the General Interface, which includes both PARCS error logicals: errcalcn, errdatan, errpvmn, and General Interface logicals: errdatagi, errpvmgi. In addition, this subroutine will check the values of TRAC-M error flags errtracm, ioerr, and jflag. The detection of any error from either the General Interface or PARCS process results in an error message being displayed indicating the origin of the error. At this point, the TDMR can begin the exit process without sending or receiving any additional data since the error-checking functionality of the other two processes should have led them to exit cleanly. If an error originates during the TRAC-M calculation, the TDMR will display the appropriate message and send indication of a thermal-hydraulic calculation fault to the General Interface process, and consequently the PARCS process, using the thermal-hydraulic logical buffer. Once the buffer has been sent, the TDMR can begin the exit process, followed by TRAC-M terminating the calculation with an error message.

II.G. Exit Handling
Exit handling in the TDMR and TRAC-M is invoked upon receipt of either a calculation- or fault based signal. A fault-based signal results from a fatal error detected in the error-checking module, as discussed in the previous section, and requires that all of the processes exit unaided, prematurely, and safely. If the error occurred either in the TDMR or TRAC-M, then the TDMR communicates the error to the General Interface process, and consequently the PARCS process, using the next available control buffer send (i.e., the next call to \texttt{TDMR}()). Once the error has been communicated, the TDMR initiates its exit procedure, which is described below.

A calculation-based signal (\texttt{tdmrdone} or \texttt{pdmrdone}) is controlled by the TRAC-M process and is transferred to the TDMR through the neutronic logical control buffer. Thus, if the second unit of the TDMR detects that \texttt{tdmrdone} is true, it will pass this information to the General Interface, and subsequently PARCS, and begin initiating its exit procedure. Upon return, the TDMR will return control to TRAC-M, which will then terminate execution as well. This procedure can be inferred from the bottom of Figure 1 in Section II.B.

Once termination of the TRAC-M / TDMR process has been determined, the subroutines \texttt{TDMRClean()} and \texttt{TDMRExit()} are called. \texttt{TDMRClean()} frees up any memory not previously de-allocated, and \texttt{TDMRExit()} removes the TRAC-M process from PVM. The coding which illustrates this procedure for subroutine \texttt{TDMR()} was shown in Section II.B.

II.H. Input Processing for the TDMR

The TDMR itself requires no input files, as the mapping information file \texttt{MAPTAB} is read by the PDMR. The mapping information in this file is functionally equivalent to its implementation in the coupled RELAP5/PARCS code, except that the RELAP5 volume and heat structure numbers are replaced by component numbers, axial levels, etc. Composite ID numbers are then formed in the PDMR for simplicity in passing the mapping information. A complete description of the composite ID number is given in Section II.A.1 of the SRS for the TDMR.
III. Summary

This document described the software design necessary to satisfy the requirement specification for the TRAC-M-Specific Data Map Routine. The major components outlined here provide the basis for code development and relate to the variable requirements, the design of the functional units, and the process flow through the TRAC-M/TDMR code. In addition, the design of the input and error-checking procedures were described and are consistent with the specified requirements.

The TRAC-M/TDMR code is designed as an independent process, and the incorporation within the framework of the coupled TRAC-M/PARCS code requires the use of both the General Interface and a message-passing protocol. The latter utilizes the PVM package to control the communication between the separate processes. The internal flow control through the functional units of the TDMR is treated logically and entry into these units is controlled by TRAC-M, as described in the previous sections.

As mentioned, the design of the TDMR is specific to the TRAC-M/PARCS coupling, and as such requires that the data transferred between TRAC-M and PARCS be well-defined. The Software Requirements Specification for both the PDMR and TDMR provided this basis, and this document has described the procedure necessary to achieve the required data coherency. Finally, the design and implementation described by this document is consistent with that for the General Interface.
IV. References


Appendix A: Variable Description

The following tables provide a description of the variables used by the TRAC-M-Specific Data Map Routine modules:

Table 1: PVM and Process Control Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Dimension</th>
<th>Description</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>tdmrdone</td>
<td>logical</td>
<td>1</td>
<td>logical flag: execution is stopped when tdmrdone = .TRUE.</td>
<td>.TRUE./.FALSE.</td>
</tr>
<tr>
<td>pdmrdone</td>
<td>logical</td>
<td>1</td>
<td>logical flag: execution is stopped when pdmrdone = .TRUE.</td>
<td>.TRUE./.FALSE.</td>
</tr>
<tr>
<td>ioutp</td>
<td>int*4</td>
<td>1</td>
<td>unit number for output</td>
<td>0 &lt; i &lt; 100</td>
</tr>
<tr>
<td>ntasks</td>
<td>int*4</td>
<td>1</td>
<td>number of processes</td>
<td>i = 3</td>
</tr>
<tr>
<td>group</td>
<td>char*6</td>
<td>1</td>
<td>6-character descriptor for the group of processes</td>
<td>“procs”</td>
</tr>
<tr>
<td>tidth</td>
<td>int*4</td>
<td>1</td>
<td>process ID for TRAC-M module</td>
<td>0 &lt; i &lt; 2^{32}</td>
</tr>
<tr>
<td>tidgi</td>
<td>int*4</td>
<td>1</td>
<td>process ID for the General Interface module</td>
<td>0 &lt; i &lt; 2^{32}</td>
</tr>
<tr>
<td>mtypeth</td>
<td>int*4</td>
<td>1</td>
<td>message tag associated with the TRAC-M module</td>
<td>i = 2</td>
</tr>
<tr>
<td>mtypegi</td>
<td>int*4</td>
<td>1</td>
<td>message tag associated with the General Interface module</td>
<td>i = 1</td>
</tr>
<tr>
<td>inum</td>
<td>int*4</td>
<td>1</td>
<td>instance number in group</td>
<td>0 ≤ i ≤ 2</td>
</tr>
<tr>
<td>istride</td>
<td>int*4</td>
<td>1</td>
<td>striding of data in the buffer</td>
<td>0 &lt; i &lt; 2^{32}</td>
</tr>
<tr>
<td>info^(a)</td>
<td>int*4</td>
<td>1</td>
<td>integer error flag for PVM calls</td>
<td>-(2^{32}) &lt; i &lt; 2^{32}</td>
</tr>
<tr>
<td>timeout</td>
<td>real*8</td>
<td>1</td>
<td>int*4 parameter indicating the max # of seconds to wait on a receive</td>
<td>i = 600</td>
</tr>
<tr>
<td>dimbuf</td>
<td>int*4</td>
<td>1</td>
<td>dimension of each of the 6 data type-dependent control buffers, in order, as shown in Tables 2 and 3</td>
<td>0 ≤ i ≤ 2^{32}</td>
</tr>
</tbody>
</table>

(a) Values less than zero indicate an error has occurred in a PVM call.
Table 2: PARCS Data and Control Buffers

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Dimension</th>
<th>Description</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>cbufn(a)</td>
<td>char*6</td>
<td>dimbuf(1)</td>
<td>control buffer of 6-character words</td>
<td>N/A</td>
</tr>
<tr>
<td>lbufn(a)</td>
<td>logical</td>
<td>dimbuf(2)</td>
<td>control buffer of logical words</td>
<td>.TRUE./.FALSE.</td>
</tr>
<tr>
<td>i2bufn(a)</td>
<td>int*2</td>
<td>dimbuf(3)</td>
<td>control buffer of 16-bit integer words</td>
<td>$-2^{16} &lt; i &lt; 2^{16}$</td>
</tr>
<tr>
<td>i4bufn(a)</td>
<td>int*4</td>
<td>dimbuf(4)</td>
<td>control buffer of 32-bit integer words</td>
<td>$-2^{32} &lt; i &lt; 2^{32}$</td>
</tr>
<tr>
<td>r4bufn(a)</td>
<td>real*4</td>
<td>dimbuf(5)</td>
<td>control buffer of 32-bit floating point words</td>
<td>$-10^{38} &lt; x &lt; 10^{38}$</td>
</tr>
<tr>
<td>r8bufn(a)</td>
<td>real*8</td>
<td>dimbuf(6)</td>
<td>control buffer of 64-bit floating point words</td>
<td>$-10^{308} &lt; x &lt; 10^{308}$</td>
</tr>
<tr>
<td>nvecnp</td>
<td>int*4</td>
<td>1</td>
<td>dimension of vecnp received from General Interface process</td>
<td>$0 &lt; i &lt; 2^{32}$</td>
</tr>
<tr>
<td>vecnp</td>
<td>real*8</td>
<td>nvecnp</td>
<td>permuted vector of space-dependent neutronic data</td>
<td>$0 \leq x \leq 10^{308}$ (b)</td>
</tr>
<tr>
<td>recvn</td>
<td>logical</td>
<td>1</td>
<td>indication the both permutation matrices are sent from the PARCS process</td>
<td>.TRUE./.FALSE.</td>
</tr>
</tbody>
</table>

(a) Used for both initial and time-dependent control information.

(b) Individual ranges are set for each component of the vector as described in Section II.F.
Table 3: TRAC-M Data and Control Buffers

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Dimension</th>
<th>Description</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>cbufth(a)</td>
<td>char*6</td>
<td>dimbuf(1)</td>
<td>control buffer of 6-character words</td>
<td>N/A</td>
</tr>
<tr>
<td>lbufth(a)</td>
<td>logical</td>
<td>dimbuf(2)</td>
<td>control buffer of logical words</td>
<td>.TRUE./.FALSE.</td>
</tr>
<tr>
<td>i2bufth(a)</td>
<td>int*2</td>
<td>dimbuf(3)</td>
<td>control buffer of 16-bit integer words</td>
<td>$-(2^{16}) &lt; i &lt; 2^{16}$</td>
</tr>
<tr>
<td>i4bufth(a)</td>
<td>int*4</td>
<td>dimbuf(4)</td>
<td>control buffer of 32-bin integer words</td>
<td>$-(2^{32}) &lt; i &lt; 2^{32}$</td>
</tr>
<tr>
<td>r4bufth(a)</td>
<td>real*4</td>
<td>dimbuf(5)</td>
<td>control buffer of 32-bit floating point words</td>
<td>$-(10^{38}) &lt; x &lt; 10^{38}$</td>
</tr>
<tr>
<td>r8bufth(a)</td>
<td>real*8</td>
<td>dimbuf(6)</td>
<td>control buffer of 64-bit floating point words</td>
<td>$-(10^{308}) &lt; x &lt; 10^{308}$</td>
</tr>
<tr>
<td>nvecth</td>
<td>int*4</td>
<td>1</td>
<td>dimension of vecth sent to General Interface process</td>
<td>$0 &lt; i &lt; 2^{32}$ (b)</td>
</tr>
<tr>
<td>vecht</td>
<td>real*8</td>
<td>nvecrth</td>
<td>vector of space-dependent thermal-hydraulic data</td>
<td></td>
</tr>
<tr>
<td>recvth</td>
<td>int*4</td>
<td>1</td>
<td>indication that both permutation matrices are sent from the TRAC-M process</td>
<td>.TRUE./.FALSE.</td>
</tr>
<tr>
<td>nodeareas</td>
<td>real*8</td>
<td>nrn</td>
<td>cross-sectional areas of each radial node in an axial level</td>
<td>$0 \leq x &lt; 10^{308}$</td>
</tr>
<tr>
<td>nodevols</td>
<td>real*8</td>
<td>nrn</td>
<td>conduction volumes of each radial node in an axial level</td>
<td>$0 \leq x &lt; 10^{308}$</td>
</tr>
<tr>
<td>wtdareas</td>
<td>real*8</td>
<td>nrn</td>
<td>weighted-areas for each radial node in an axial level</td>
<td>$0 \leq x &lt; 10^{308}$</td>
</tr>
<tr>
<td>mycqs</td>
<td>real*8</td>
<td>nrn</td>
<td>volume fraction of solid material for each radial node</td>
<td>$0 \leq x &lt; 10^{308}$</td>
</tr>
<tr>
<td>temprpowf</td>
<td>real*8</td>
<td>nrn</td>
<td>weighted radial power distribution within an axial level</td>
<td>$0 \leq x &lt; 10^{308}$</td>
</tr>
<tr>
<td>compptrs</td>
<td>int*4</td>
<td>numvols</td>
<td>pointers to thermal-hydraulic component array indices</td>
<td>$0 &lt; i \leq 999$</td>
</tr>
</tbody>
</table>
Table 3: TRAC-M Data and Control Buffers

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Dimension</th>
<th>Description</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>iptrs</td>
<td>int*4</td>
<td>numvols</td>
<td>pointers to the first dimensions within thermal-hydraulic arrays</td>
<td>$0 &lt; i &lt; 2^{32}$</td>
</tr>
<tr>
<td>jptrs</td>
<td>int*4</td>
<td>numvols</td>
<td>pointers to the second dimension within thermal-hydraulic arrays</td>
<td>$0 &lt; i &lt; 2^{32}$</td>
</tr>
<tr>
<td>kptrs</td>
<td>int*4</td>
<td>numvols</td>
<td>pointers to the third dimension within thermal-hydraulic arrays</td>
<td>$0 &lt; i &lt; 2^{32}$</td>
</tr>
<tr>
<td>is3d</td>
<td>logical</td>
<td>numvols</td>
<td>indication of whether thermal-hydraulic component is three-dimensional VESSEL</td>
<td>.TRUE./.FALSE.</td>
</tr>
<tr>
<td>hspros</td>
<td>int*4</td>
<td>numhss</td>
<td>pointers to heat structure component array indices</td>
<td>$0 &lt; i &lt; 999$</td>
</tr>
<tr>
<td>hsrods</td>
<td>int*4</td>
<td>numhss</td>
<td>pointers to the average ROD/SLAB within the heat structure component</td>
<td>$0 &lt; i &lt; 2^{32}$</td>
</tr>
<tr>
<td>hsaxial</td>
<td>int*4</td>
<td>numhss</td>
<td>pointers to the axial level within the average ROD/SLAB</td>
<td>$0 &lt; i &lt; 2^{32}$</td>
</tr>
<tr>
<td>hsfsn</td>
<td>int*4</td>
<td>numhss</td>
<td>pointers to the fuel surface node within the axial level</td>
<td>$0 &lt; i &lt; 2^{32}$</td>
</tr>
<tr>
<td>isslab</td>
<td>logical</td>
<td>numhss</td>
<td>indication of whether heat structure component is rectangular SLAB</td>
<td>.TRUE./.FALSE.</td>
</tr>
<tr>
<td>ischanth</td>
<td>logical</td>
<td>numvols</td>
<td>indication of whether PIPE is actually a CHAN</td>
<td>.TRUE./.FALSE.</td>
</tr>
<tr>
<td>ischanhs</td>
<td>logical</td>
<td>numhss</td>
<td>indication of whether ROD is part of a CHAN</td>
<td>.TRUE./.FALSE.</td>
</tr>
</tbody>
</table>

(a) Used for both initial and time-dependent control information.
(b) Individual ranges are set for each component of the vector as described in Section II.F.
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Dimension</th>
<th>Description</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>nsubth2n</td>
<td>int*4</td>
<td>1</td>
<td>parameter identifying the number of thermal-hydraulic-to-neutronic variables</td>
<td>i = 5</td>
</tr>
<tr>
<td>nsubhs2n</td>
<td>int*4</td>
<td>1</td>
<td>parameter identifying the number of heat structure-to-neutronic variables</td>
<td>i = 3</td>
</tr>
<tr>
<td>nsubn2th</td>
<td>int*4</td>
<td>1</td>
<td>parameter identifying the number of neutronic-to-thermal-hydraulic variables</td>
<td>i = 1</td>
</tr>
<tr>
<td>nsubn2hs</td>
<td>int*4</td>
<td>1</td>
<td>parameter identifying the number of neutronic-to-heat structure variables</td>
<td>i = 1</td>
</tr>
<tr>
<td>numvols</td>
<td>int*4</td>
<td>1</td>
<td>number of thermal-hydraulic cells</td>
<td>0 ≤ i &lt; 2^{32}</td>
</tr>
<tr>
<td>numhss</td>
<td>int*4</td>
<td>1</td>
<td>number of heat structure axial nodes</td>
<td>0 ≤ i &lt; 2^{32}</td>
</tr>
<tr>
<td>ncolth2n</td>
<td>int*4</td>
<td>1</td>
<td>total number of columns in thermal-hydraulic/heat structure to neutronic matrix</td>
<td>i = max(matcolth)</td>
</tr>
<tr>
<td>nrown2th</td>
<td>int*4</td>
<td>1</td>
<td>total number of rows in neutronic to thermal-hydraulic/heat structure matrix</td>
<td>i = max(matrown)</td>
</tr>
<tr>
<td>ivolndx</td>
<td>int*4</td>
<td>numvols</td>
<td>array of thermal-hydraulic composite ID numbers</td>
<td>0 ≤ i &lt; 2^{32}</td>
</tr>
<tr>
<td>ihsndx</td>
<td>int*4</td>
<td>numhss</td>
<td>array of heat structure composite ID numbers</td>
<td>0 ≤ i &lt; 2^{32}</td>
</tr>
</tbody>
</table>
### Table 5: Miscellaneous Calculational Control Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Dimension</th>
<th>Description</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>itrip</td>
<td>int*4</td>
<td>1</td>
<td>TRAC-M trip ID number (user input)</td>
<td>$0 &lt; i &lt; 10000$</td>
</tr>
<tr>
<td>scramtrip</td>
<td>logical</td>
<td>1</td>
<td>status of trip #itrip</td>
<td>.TRUE./.FALSE.</td>
</tr>
<tr>
<td>rstrt</td>
<td>logical</td>
<td>1</td>
<td>indication of whether the current calculation is a restart</td>
<td>.TRUE./.FALSE.</td>
</tr>
<tr>
<td>majed</td>
<td>logical</td>
<td>1</td>
<td>indication of whether a major edit is to be performed</td>
<td>.TRUE./.FALSE.</td>
</tr>
<tr>
<td>rsted</td>
<td>logical</td>
<td>1</td>
<td>indication of whether a restart edit is to be performed</td>
<td>.TRUE./.FALSE.</td>
</tr>
<tr>
<td>ssinit</td>
<td>logical</td>
<td>1</td>
<td>indication of whether the current calculation is a steady-state initialization</td>
<td>.TRUE./.FALSE.</td>
</tr>
<tr>
<td>cbccalc</td>
<td>logical</td>
<td>1</td>
<td>indication of whether a boron criticality search is being performed</td>
<td>.TRUE./.FALSE.</td>
</tr>
<tr>
<td>cbcratio</td>
<td>real*8</td>
<td>1</td>
<td>boron concentration adjustment ratio from by PARCS criticality search</td>
<td>.TRUE./.FALSE.</td>
</tr>
<tr>
<td>rodnum</td>
<td>int*4</td>
<td>1</td>
<td>indication of current average ROD/SLAB within a heat structure</td>
<td>$0 &lt; i &lt; 2^{32}$</td>
</tr>
<tr>
<td>crcntl</td>
<td>logical</td>
<td>1</td>
<td>indication of whether TRAC-M is controlling rod bank positions</td>
<td>.TRUE./.FALSE.</td>
</tr>
<tr>
<td>nrcntl</td>
<td>int*4</td>
<td>1</td>
<td>number of control rod banks in PARCS controlled by TRAC-M</td>
<td>$0 &lt; i &lt; 2^{32}$</td>
</tr>
<tr>
<td>isgv</td>
<td>int*4</td>
<td>nrcntl</td>
<td>signal variable number controlling rod bank position</td>
<td>$0 &lt; i &lt; 2^{32}$</td>
</tr>
<tr>
<td>sgvbank</td>
<td>int*4</td>
<td>nrcntl</td>
<td>control rod bank number in PARCS being controlled by TRAC-M</td>
<td>$0 &lt; i &lt; 2^{32}$</td>
</tr>
<tr>
<td>flagneut</td>
<td>logical</td>
<td>1</td>
<td>flag indicating whether PARCS is tightly converged (not used)</td>
<td>.TRUE./.FALSE.</td>
</tr>
<tr>
<td>pow3d</td>
<td>real*8</td>
<td>1</td>
<td>total reactor power</td>
<td>$0 \leq x \leq 10^{308}$</td>
</tr>
<tr>
<td>rho3d</td>
<td>real*8</td>
<td>1</td>
<td>total (dynamic) reactivity</td>
<td>$-(10^{308}) \leq x \leq 10^{308}$</td>
</tr>
</tbody>
</table>
Table 5: Miscellaneous Calculational Control Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Dimension</th>
<th>Description</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>keff3d</td>
<td>real*8</td>
<td>1</td>
<td>neutron multiplication factor</td>
<td>$0 \leq x \leq 10^{308}$</td>
</tr>
<tr>
<td>initcrp</td>
<td>real*8</td>
<td>nrcntl</td>
<td>initial control rod bank insertion fractions</td>
<td>$0 \leq x \leq 1.0$</td>
</tr>
<tr>
<td>newcrp</td>
<td>real*8</td>
<td>nrcntl</td>
<td>new control rod bank insertion fractions</td>
<td>$0 \leq x \leq 1.0$</td>
</tr>
<tr>
<td>automap</td>
<td>logical</td>
<td>1</td>
<td>indication of whether the current case if “automapped”</td>
<td>.TRUE./.FALSE.</td>
</tr>
<tr>
<td>chan-groups</td>
<td>int*4</td>
<td>1</td>
<td>number of ROD groups mapped to kinetics</td>
<td>$0 &lt; i &lt; 2^{32}$</td>
</tr>
<tr>
<td>chanlevels</td>
<td>int*4</td>
<td>1</td>
<td>number of ROD axial levels mapped to kinetics</td>
<td>$0 &lt; i &lt; 2^{32}$</td>
</tr>
</tbody>
</table>

Table 6: Variables Associated with the Error Checking

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Dimension</th>
<th>Description</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>errcalcn</td>
<td>logical</td>
<td>1</td>
<td>indication of a calculation error in PARCS</td>
<td>.TRUE./.FALSE.</td>
</tr>
<tr>
<td>errdatan</td>
<td>logical</td>
<td>1</td>
<td>indication of a data error in the PDMR</td>
<td>.TRUE./.FALSE.</td>
</tr>
<tr>
<td>errpvmn</td>
<td>logical</td>
<td>1</td>
<td>indication of a PVM error in the PDMR</td>
<td>.TRUE./.FALSE.</td>
</tr>
<tr>
<td>ern</td>
<td>logical</td>
<td>1</td>
<td>indication of an error in the neutronics side</td>
<td>.TRUE./.FALSE.</td>
</tr>
<tr>
<td>errextn</td>
<td>logical</td>
<td>1</td>
<td>indication of an error in the external neutronics calcs.</td>
<td>.TRUE./.FALSE.</td>
</tr>
<tr>
<td>errcalcgi</td>
<td>logical</td>
<td>1</td>
<td>indication of a calculation error in the General Interface</td>
<td>.TRUE./.FALSE.</td>
</tr>
<tr>
<td>errdatagi</td>
<td>logical</td>
<td>1</td>
<td>indication of a data error in the General Interface</td>
<td>.TRUE./.FALSE.</td>
</tr>
<tr>
<td>errpvmgi</td>
<td>logical</td>
<td>1</td>
<td>indication of a PVM error in the General Interface</td>
<td>.TRUE./.FALSE.</td>
</tr>
<tr>
<td>Name</td>
<td>Type</td>
<td>Dimension</td>
<td>Description</td>
<td>Range</td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td>-----------</td>
<td>---------------------------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>errgi</td>
<td>logical</td>
<td>1</td>
<td>indication of an error in the General Interface</td>
<td>.TRUE./.FALSE.</td>
</tr>
<tr>
<td>errcalcth</td>
<td>logical</td>
<td>1</td>
<td>indication of a calculation error in TRAC-M</td>
<td>.TRUE./.FALSE.</td>
</tr>
<tr>
<td>errdatath</td>
<td>logical</td>
<td>1</td>
<td>indication of a data error in the TDMR</td>
<td>.TRUE./.FALSE.</td>
</tr>
<tr>
<td>errpvmth</td>
<td>logical</td>
<td>1</td>
<td>indication of a PVM error in the TDMR</td>
<td>.TRUE./.FALSE.</td>
</tr>
<tr>
<td>errinit</td>
<td>logical</td>
<td>1</td>
<td>indication of error detected in the Initialization unit of the TDMR</td>
<td>.TRUE./.FALSE.</td>
</tr>
<tr>
<td>errth2n</td>
<td>logical</td>
<td>1</td>
<td>indication of error detected in the Thermal-Hydraulic to Neutronic mapping unit of the TDMR</td>
<td>.TRUE./.FALSE.</td>
</tr>
<tr>
<td>errn2th</td>
<td>logical</td>
<td>1</td>
<td>indication of error detected in the Neutronic to Thermal-Hydraulic mapping unit of the TDMR</td>
<td>.TRUE./.FALSE.</td>
</tr>
<tr>
<td>errtracm</td>
<td>logical</td>
<td>1</td>
<td>indication of error detected in TRAC-M</td>
<td>.TRUE./.FALSE.</td>
</tr>
<tr>
<td>errtdmr</td>
<td>logical</td>
<td>1</td>
<td>indication of error detected in the TDMR</td>
<td>.TRUE./.FALSE.</td>
</tr>
<tr>
<td>Name</td>
<td>Type</td>
<td>Dimension</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>--------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>constant.h</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pi</td>
<td>real*8</td>
<td>1</td>
<td>Mathematical constant: $\pi$</td>
<td></td>
</tr>
<tr>
<td>jflag</td>
<td>int*4</td>
<td>1</td>
<td>flag for indicating input processing error</td>
<td></td>
</tr>
<tr>
<td>BreakArray</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>breakAr%contb</td>
<td>real*8</td>
<td>breakTab%nbtb</td>
<td>element of derived data type breakArT defining time-dependent boron concentration table for BREAK</td>
<td></td>
</tr>
<tr>
<td>BreakVlt</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>breakTab%ibty</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type breakTabT indicating the type of BREAK</td>
<td></td>
</tr>
<tr>
<td>breakTab%nbtb</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type breakTabT indicating the number of data pairs in breakAr%contb</td>
<td></td>
</tr>
<tr>
<td>breakTab%ibsv</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type breakTabT indicating the signal variable ID for evaluating breakAr%contb</td>
<td></td>
</tr>
<tr>
<td>breakTab%conoff</td>
<td>real*8</td>
<td>1</td>
<td>element of derived data type breakTabT defining boron concentration when controlling trip is OFF after being ON</td>
<td></td>
</tr>
<tr>
<td>ChanTab</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>chanTab%icrnk</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type chanTabT indicating the number of cells below the core neutronic region</td>
<td></td>
</tr>
<tr>
<td>CompTyp</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pipeh</td>
<td>real*8</td>
<td>1</td>
<td>parameter identifying PIPE ($x = 1.0d0$)</td>
<td></td>
</tr>
<tr>
<td>vsslh</td>
<td>real*8</td>
<td>1</td>
<td>parameter identifying VESSEL ($x = 2.0d0$)</td>
<td></td>
</tr>
<tr>
<td>rodh</td>
<td>real*8</td>
<td>1</td>
<td>parameter identifying ROD ($x = 3.0d0$)</td>
<td></td>
</tr>
<tr>
<td>slabh</td>
<td>real*8</td>
<td>1</td>
<td>parameter identifying SLAB ($x = 4.0d0$)</td>
<td></td>
</tr>
</tbody>
</table>
Table 7: TRAC-M Variables Utilized in the TDMR

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Dimension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pumph</td>
<td>real*8</td>
<td>1</td>
<td>parameter identifying PUMP (x=5.0d0)</td>
</tr>
<tr>
<td>teeh</td>
<td>real*8</td>
<td>1</td>
<td>parameter identifying TEE (x=6.0d0)</td>
</tr>
<tr>
<td>valveh</td>
<td>real*8</td>
<td>1</td>
<td>parameter identifying VALVE (x=7.0d0)</td>
</tr>
<tr>
<td>prizrh</td>
<td>real*8</td>
<td>1</td>
<td>parameter identifying PRIZR (x=8.0d0)</td>
</tr>
<tr>
<td>turbh</td>
<td>real*8</td>
<td>1</td>
<td>parameter identifying TURB (x=9.0d0)</td>
</tr>
<tr>
<td>sepdh</td>
<td>real*8</td>
<td>1</td>
<td>parameter identifying SEPD (x=10.0d0)</td>
</tr>
<tr>
<td>breakh</td>
<td>real*8</td>
<td>1</td>
<td>parameter identifying BREAK (x=11.0d0)</td>
</tr>
<tr>
<td>fillh</td>
<td>real*8</td>
<td>1</td>
<td>parameter identifying FILL (x=12.0d0)</td>
</tr>
<tr>
<td>plenh</td>
<td>real*8</td>
<td>1</td>
<td>parameter identifying PLENUM (x=13.0d0)</td>
</tr>
<tr>
<td>chanh</td>
<td>real*8</td>
<td>1</td>
<td>parameter identifying CHAN (x=15.0d0)</td>
</tr>
<tr>
<td>CSDat</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>csGl%ntrp</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type csGlt indicating number of trips defined in input</td>
</tr>
<tr>
<td>csGl%ntsv</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type csGlt indicating number of signal variables defined in input</td>
</tr>
<tr>
<td>csTrip%idtp</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type csTripT indicating the trip ID number</td>
</tr>
<tr>
<td>csSig%idsv</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type csSigT indicating the signal variable ID number</td>
</tr>
<tr>
<td>csSig%ilcn1</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type csSigT indicating the PARCS rod bank number</td>
</tr>
<tr>
<td>FillArray</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fillAr%contb</td>
<td>real*8</td>
<td>fill-Tab%nftb</td>
<td>element of derived data type fillArT defining time-dependent boron concentration table for FILL</td>
</tr>
<tr>
<td>FillVlt</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fillTab%ifty</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type fillTabT indicating the type of FILL</td>
</tr>
<tr>
<td>fillTab%nftb</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type fillTabT indicating the number of data pairs in fillAr%contb</td>
</tr>
<tr>
<td>Name</td>
<td>Type</td>
<td>Dimension</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>----------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>fillTab%ifsv</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type fillTabT indicating the signal variable ID for evaluating fillAr%contb</td>
</tr>
<tr>
<td>fillTab%conoff</td>
<td>real*8</td>
<td>1</td>
<td>element of derived data type fillTabT defining boron concentration when controlling trip is OFF after being ON</td>
</tr>
<tr>
<td>genTab%type</td>
<td>real*8</td>
<td>1</td>
<td>element of derived data type genTabT identifying component type</td>
</tr>
<tr>
<td>genTab%num</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type genTabT indicating component number</td>
</tr>
<tr>
<td>genTab%nodes</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type genTabT indicating the number of heat conduction nodes</td>
</tr>
<tr>
<td>g1DAr%dx</td>
<td>real*8</td>
<td>ncells</td>
<td>element of derived data type g1DArrayT storing lengths of cells</td>
</tr>
<tr>
<td>g1DAr%tln</td>
<td>real*8</td>
<td>ncells</td>
<td>element of derived data type g1DArrayT storing new liquid temperature</td>
</tr>
<tr>
<td>g1DAr%roln</td>
<td>real*8</td>
<td>ncells</td>
<td>element of derived data type g1DArrayT storing new liquid density</td>
</tr>
<tr>
<td>g1DAr%rovn</td>
<td>real*8</td>
<td>ncells</td>
<td>element of derived data type g1DArrayT storing new vapor density</td>
</tr>
<tr>
<td>g1DAr%alpn</td>
<td>real*8</td>
<td>ncells</td>
<td>element of derived data type g1DArrayT storing new void fraction</td>
</tr>
<tr>
<td>g1DAr%conc</td>
<td>real*8</td>
<td>ncells</td>
<td>element of derived data type g1DArrayT storing old solute concentration</td>
</tr>
<tr>
<td>g1DAr%concen</td>
<td>real*8</td>
<td>ncells</td>
<td>element of derived data type g1DArrayT storing new solute concentration</td>
</tr>
<tr>
<td>compIndices</td>
<td>int*4</td>
<td>maxComps</td>
<td>pointers to components within TRAC-M</td>
</tr>
<tr>
<td>timet</td>
<td>real*8</td>
<td>1</td>
<td>computation time</td>
</tr>
</tbody>
</table>

Table 7: TRAC-M Variables Utilized in the TDMR
Table 7: TRAC-M Variables Utilized in the TDMR

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Dimension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>delt</td>
<td>real*8</td>
<td>1</td>
<td>current timestep size</td>
</tr>
<tr>
<td>odelt</td>
<td>real*8</td>
<td>1</td>
<td>old timestep size</td>
</tr>
<tr>
<td>isolut</td>
<td>int*4</td>
<td>1</td>
<td>indication of whether solute tracking is on</td>
</tr>
<tr>
<td>stdyst</td>
<td>int*4</td>
<td>1</td>
<td>indication of type of steady-state calculation</td>
</tr>
<tr>
<td>itdmr</td>
<td>int*4</td>
<td>1</td>
<td>indication of whether to use 3-D kinetics</td>
</tr>
<tr>
<td>itmrp</td>
<td>int*4</td>
<td>1</td>
<td>indication of verbosity of TDMR edits</td>
</tr>
<tr>
<td>nchans</td>
<td>int*4</td>
<td>1</td>
<td>indication of number of CHAN</td>
</tr>
</tbody>
</table>

GlobalDim

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Dimension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ncomp</td>
<td>int*4</td>
<td>1</td>
<td>number of thermal-hydraulic components in TRAC-M model</td>
</tr>
<tr>
<td>nhtstr</td>
<td>int*4</td>
<td>1</td>
<td>number of heat structures in TRAC-M model</td>
</tr>
<tr>
<td>ncompt</td>
<td>int*4</td>
<td>1</td>
<td>number of thermal-hydraulic and heat structures in TRAC-M model</td>
</tr>
</tbody>
</table>

HSArray

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Dimension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hsAr%rdz</td>
<td>real*8</td>
<td>(ncr,ncrz+1, nodes)</td>
<td>element of derived data type hsArT storing axial level heights</td>
</tr>
<tr>
<td>hsAr%radrn</td>
<td>real*8</td>
<td>(ncr,ncrz+1, nodes)</td>
<td>element of derived data type hsArT storing new radial node radii</td>
</tr>
<tr>
<td>hsAr%rftn</td>
<td>real*8</td>
<td>(ncr,ncrz+1, nodes)</td>
<td>element of derived data type hsArT storing new fuel temperature</td>
</tr>
<tr>
<td>hsAr%rdpwr</td>
<td>real*8</td>
<td>nodes</td>
<td>element of derived data type hsArT storing radial power distribution</td>
</tr>
<tr>
<td>hsAr%hcomo</td>
<td>int*4</td>
<td>ncrz+1</td>
<td>element of derived data type hsArT storing the component number coupled with the HS</td>
</tr>
<tr>
<td>hsAr%hcelo</td>
<td>int*4</td>
<td>ncrz+1</td>
<td>element of derived data type hsArT storing the cell number coupled with the HS</td>
</tr>
<tr>
<td>hsAr%cpowr</td>
<td>real*8</td>
<td>ncrx</td>
<td>element of derived data type hsArT storing the peaking factors for each ROD group</td>
</tr>
<tr>
<td>hsAr%nrdx</td>
<td>real*8</td>
<td>ncrx</td>
<td>element of derived data type hsArT storing the number of actual fuel pins in HS group</td>
</tr>
<tr>
<td>Name</td>
<td>Type</td>
<td>Dimension</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>--------</td>
<td>-----------</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td>sik</td>
<td>int*4</td>
<td>1</td>
<td>defines standard integer KIND</td>
</tr>
<tr>
<td>sdk</td>
<td>int*4</td>
<td>1</td>
<td>defines standard double precision KIND</td>
</tr>
<tr>
<td>itty</td>
<td>int*4</td>
<td>1</td>
<td>file handle indicating standard terminal output</td>
</tr>
<tr>
<td>iout</td>
<td>int*4</td>
<td>1</td>
<td>file handle indicating TRCOUT</td>
</tr>
<tr>
<td>imout</td>
<td>int*4</td>
<td>1</td>
<td>file handle indicating TRCMSG</td>
</tr>
<tr>
<td>pipe-Tab%ncells</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type pipeTabT defining the number of cells making up this PIPE</td>
</tr>
<tr>
<td>pipeTab%iconc</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type pipeTabT defining whether solute is present in this PIPE</td>
</tr>
<tr>
<td>pipeTab%cpow</td>
<td>real*8</td>
<td>1</td>
<td>element of derived data type pipeTabT storing total energy deposited into the PIPE coolant for the current time step</td>
</tr>
<tr>
<td>pipe-Tab%eninp</td>
<td>real*8</td>
<td>1</td>
<td>element of derived data type pipeTabT storing cumulative energy deposition into the coolant of the PIPE</td>
</tr>
<tr>
<td>pipeTab%ipow</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type pipeTabT indicating whether direct coolant heating option is activated for this PIPE</td>
</tr>
<tr>
<td>plenTab%iconc</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type plenTabT defining whether solute is present in this PLENUM</td>
</tr>
<tr>
<td>prizeTab%ncells</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type prizeTabT defining the number of cells making up this PRIZR</td>
</tr>
<tr>
<td>prizeTab%iconc</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type prizeTabT defining whether solute is present in this PRIZR</td>
</tr>
</tbody>
</table>
Table 7: TRAC-M Variables Utilized in the TDMR

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Dimension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pumpTab%ncells</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type pumpTabT defining the number of cells making up this PUMP</td>
</tr>
<tr>
<td>pumpTab%conc</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type pumpTabT defining whether solute is present in this PUMP</td>
</tr>
<tr>
<td>RodVlt</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rodTab%irpwty</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type rodTabT storing the option for reactivity or power type</td>
</tr>
<tr>
<td>rodTab%ncrz</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type rodTabT storing the number of axial intervals in this heat structure</td>
</tr>
<tr>
<td>rod-Tab%rpowrn</td>
<td>real*8</td>
<td>1</td>
<td>element of derived data type rodTabT storing end of timestep reactor core power</td>
</tr>
<tr>
<td>rod-Tab%power</td>
<td>real*8</td>
<td>1</td>
<td>element of derived data type rodTabT storing average reactor core power over the timestep</td>
</tr>
<tr>
<td>rodTab%width</td>
<td>real*8</td>
<td>1</td>
<td>element of derived data type rodTabT storing width of the SLAB surface</td>
</tr>
<tr>
<td>rodTab%nopowr</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type rodTabT indicating whether no heat source is present</td>
</tr>
<tr>
<td>rodTab%irftr</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type rodTabT indicating trip ID controlling axial refinement</td>
</tr>
<tr>
<td>rodTab%ncrx</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type rodTabT indicating the number of average ROD/SLAB elements in this heat structure</td>
</tr>
<tr>
<td>TeeVlt</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>teeTab%iconc1</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type teeTabT defining whether solute is present in the first section of this TEE</td>
</tr>
<tr>
<td>teeTab%iconc2</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type teeTabT defining whether solute is present in the second section of this TEE</td>
</tr>
</tbody>
</table>
Table 7: TRAC-M Variables Utilized in the TDMR

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Dimension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tteTab%iconc</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type tteTabT defining the number of cells making up this TEE</td>
</tr>
<tr>
<td>valveTab%ncells</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type valveTabT defining the number of cells making up this VALVE</td>
</tr>
<tr>
<td>valveTab%iconc</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type valveTabT defining whether solute is present in this VALVE</td>
</tr>
<tr>
<td>vsAr3%tln</td>
<td>real*8</td>
<td>(nrsx,ntsx,nasx)</td>
<td>element of derived data type vessArray3T storing new liquid temperature</td>
</tr>
<tr>
<td>vsAr3%roln</td>
<td>real*8</td>
<td>(nrsx,ntsx,nasx)</td>
<td>element of derived data type vessArray3T storing new liquid density</td>
</tr>
<tr>
<td>vsAr3%rovn</td>
<td>real*8</td>
<td>(nrsx,ntsx,nasx)</td>
<td>element of derived data type vessArray3T storing new vapor density</td>
</tr>
<tr>
<td>vsAr3%alpn</td>
<td>real*8</td>
<td>(nrsx,ntsx,nasx)</td>
<td>element of derived data type vessArray3T storing new void fraction</td>
</tr>
<tr>
<td>vsAr3%conc</td>
<td>real*8</td>
<td>(nrsx,ntsx,nasx)</td>
<td>element of derived data type vessArray3T storing new solute concentration</td>
</tr>
<tr>
<td>vsAr3%q3drl</td>
<td>real*8</td>
<td>(nrsx,ntsx,nasx)</td>
<td>element of derived data type vessArray3T storing direct moderator power to liquid</td>
</tr>
<tr>
<td>vsAr3%q3drv</td>
<td>real*8</td>
<td>(nrsx,ntsx,nasx)</td>
<td>element of derived data type vessArray3T storing direct moderator power to vapor</td>
</tr>
<tr>
<td>vessTab%iconc</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type vessTabT indicating whether solute is present in this VESSEL</td>
</tr>
<tr>
<td>vessTab%ic0m</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type vessTabT storing the array starting location for radial data in the vsAr3 arrays</td>
</tr>
<tr>
<td>vessTab%ic0m</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type vessTabT storing offset information for radial vsAr3 data</td>
</tr>
</tbody>
</table>
Table 7: TRAC-M Variables Utilized in the TDMR

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Dimension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vessTab%jc0m</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type vessTabT storing the array starting location for sector data in the vsAr3 arrays</td>
</tr>
<tr>
<td>vessTab%jc0m</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type vessTabT storing offset information for sector vsAr3 data</td>
</tr>
<tr>
<td>vessTab%kc0mm</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type vessTabT storing the array starting location for axial data in the vsAr3 arrays</td>
</tr>
<tr>
<td>vessTab%kc0mm</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type vessTabT storing offset information for axial vsAr3 data</td>
</tr>
<tr>
<td>vessTab%iall</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type vessTabT storing the last element of radial data in the vsAr3 data</td>
</tr>
<tr>
<td>vessTab%jall</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type vessTabT storing the last element of sector data in the vsAr3 data</td>
</tr>
<tr>
<td>vessTab%kall</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type vessTabT storing the last element of axial data in the vsAr3 data</td>
</tr>
<tr>
<td>vessTab%nrsx</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type vessTabT storing the number of radial rings</td>
</tr>
<tr>
<td>vessTab%ntsx</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type vessTabT storing the number of theta sectors</td>
</tr>
<tr>
<td>vessTab%nasx</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type vessTabT storing the number of axial levels</td>
</tr>
<tr>
<td>vessTab%nclx</td>
<td>int*4</td>
<td>1</td>
<td>element of derived data type vessTabT storing the number of fluid cells per axial level</td>
</tr>
</tbody>
</table>
Appendix B: Subroutine Report

**Subroutines Deleted:**

none

**Subroutines Added:**

Subroutine TDMR: Logic Controller for the TRAC-M-Specific Data Map Routine.
Uses Modules: IntrType, Io, TDMRVarDecl

Contains Subroutines:
  - Subroutine TDMRInit(): Controls the initialization functional unit.
    Uses Modules: TDMRErrorCheck, TDMRInitCalc
  - Subroutine TDMRth2n: Controls the Thermal-Hydraulic to Neutronic Mapping.
    Uses Modules: GlobalDat, TDMRErrorCheck, TDMRTimeCalc
  - Subroutine TDMRn2th: Controls the Neutronic to Thermal-Hydraulic Mapping.
    Uses Modules: GlobalDat, TDMRErrorCheck, TDMRTimeCalc
  - Subroutine TDMRClean: Frees up memory for arrays not previously deallocated.
    Uses Modules: GlobalDat
  - Subroutine TDMRExit: Removes TRAC-M process from PVM.

Module TDMRInitCalc: Contains the subroutines used during the initialization stage.
Uses Modules: Io, TDMRErrorCheck, TDMRVarDecl

Contains Subroutines:
  - Subroutine TDMRObtainIDs: Establishes communications with the General Interface process.

  Subroutine TDMRBufInit: Communicates initial control buffers with the General Interface process.
  Uses Modules: BadInput, ChanVlt, CorePowerAr, Flt, Gen1DArray, GlobalDat, HSArray, IntrType, PipeVlt, RodVlt, TDMRCommBuf, VessArray3, VessVlt
Subroutines Added (cont.):

Module TDMRTimeCalc:  Contains the subroutines used for the time-
dependent data mapping.
Uses Modules:  TDMRVarDecl

Contains Subroutines:
Subroutine TDMRBufth2n:  Sends thermal-hydraulic control
buffer to the General Interface process.
Uses Modules:  CSEval, IntrType

Subroutines TDMRBufn2th:  Receives neutronic control
buffer from the General Interface process
Uses Modules:  TDMRCommBuf

Subroutine TDMRMapth2n:  Builds the vector of TRAC-M
thermal-hydraulic data and sends the vector to the
General Interface process.
Uses Modules:  Gen1DArray, GlobalDat, HSArray,
IntrType, Io, RodVlt, TDMRCommBuf, TDMRErrorCheck,
VessArray3

Subroutine TDMRMapn2th:  Receives permuted vector of
neutronic data from the General Interface process
and stores in appropriate TRAC-M memory locations.
Uses Modules:  IntrType, CorePowerDat, GlobalDat,
PipeVlt, TDMRErrorCheck, RodVlt, VessArray3

Subroutine TDMRAdjBoron:  Controls the adjustment of
boron for boron criticality searches.
Uses Modules:  BreakArray, BreakVlt, CompTyp,
FillArray, FillVlt, Flt, Gen1DArray, Global,
GlobalDim, IntrType, Io, PipeVlt, PlenVlt,
PrizeVlt, PumpVlt, TeeVlt, TextIo, ValveVlt,
VessArray3, VessVlt

Subroutine TDMRPipeCPOW:  Controls the amount of power
deposited into the coolant for PIPE components.
Uses Modules:  GlobalDat, IntrType, PipeVlt

Subroutine TDMRPipeQ:  Controls the distribution of power
deposited into the coolant for each PIPE cell.
Uses Modules:  IntrType

Subroutine TDMRHtstrQppp:  Controls calculation of
volumetric heat generation rates for heat structures.
Uses Modules:  Global, IntrType, RodVlt
Subroutines Added (cont.):

Module TDMRCommBuf: Contains the subroutines used to process and communicate control buffers with the General Interface process.
Uses Modules: IntrType, TDMRErrorCheck, TDMRVarDecl
Contains Subroutines:
Subroutine TDMRRecvBufn: Receive neutronic control buffer from the General Interface process.

Subroutine TDMRSendBufth: Sends thermal-hydraulic control buffer to the General Interface process.

Module TDMRErrorCheck: Contains the subroutines used to perform the error checking for the TRAC-M-Specific Data Map Routine.
Uses Modules: IntrType, Io, TDMRVarDecl
Contains Subroutines:
Subroutine TDMRDataErrChk: Performs error checking on data used by TRAC-M-Specific Data Map Routine.
Uses Modules: CompTyp, CSDat, Flt, Global, GlobalDim, HSArray, PipeVlt, RodVlt, VessVlt
Subroutine TDMRPVMErrChk: Performs error checking on PVM status variables.
Subroutine TDMRProcErrChk: Check the value of error logicals sent from the General Interface which relate to both neutronic and General Interface errors.
Uses Modules: BadInput

Module TDMRVarDecl: Declares the variables used by the TRAC-M-Specific Data Map Routine.
Uses Modules: IntrType

Subroutines (TRAC-M) Modified:

TracDumpM.f90 (dmpit)- added code to set rsted=.TRUE. when TRAC-M performs a restart edit; added use of TDMRVarDecl for access to rsted
edit.f90 - added code to set majed=.TRUE. when TRAC-M performs a major edit; added use of TDMRVarDecl for access to majed
error.f90 - added use of TDMRVarDecl for access to errtdmr and errtracm; added use of BadInput for access to jflag; added
logic for setting errtracm=.TRUE. and calling TDMR(1) if input processing error occurred; added logic for setting errtracm=.TRUE. and calling TDMR(2) to shut down TDMR/GI/PARCS if TRAC-M called this subroutine; added logic for terminating TRAC-M if TDMR called this subroutine; added logic for setting errtracm=.TRUE. and calling TDMR(2) if steady-state calculation failed to converge

Gen1DCrunchM.f90 (tf1ds) - added use of TDMRTimeCalc for access to subroutine TDMRPipeQ; added LOGICAL coupled for indicating whether current component is coupled to kinetics; added logic to route calculational flow through TDMRPipeQ if coupled calculation is being performed

GlobalDatM.f90 - added itdmr and default value of 0 to indicate whether to use the calls to the TRAC-M-Specific Data Map Routine; added itmrp and default value of 0 to indicate the verbosity with which the TRAC-M-Specific Data Map Routine will operate

PipeM.f90 (pipe1) - added use of TDMRTimeCalc for access to subroutine TDMRPipeCPOW; added LOGICAL coupled for indicating whether current component is coupled to kinetics; added logic to route calculational flow through TDMRPipeCPOW if coupled calculation is being performed

TracDumpM.f90 (rdrest) - added use of TDMRVarDecl for access to rstrt; added code to set rstrt based on whether TCRST is opened

RodCrunchM.f90 (rodht) - added use of TDMRTimeCalc for access to subroutine TDMRHtstrQppp; added LOGICAL coupled for indicating whether current component is coupled to kinetics; added logic to route calculational flow through TDMRHtstrQppp if coupled calculation is being performed

steady.f90 - added logic to route calculational flow through TDMR(2) and TDMR(3) if current calculation is coupled to kinetics (itdmr=1)

trac.f90 - added use of TDMRVarDecl for access to tdmrdone; added code to call TDMR(1), TDMR(2), and TDMR(3) before first restart dump is performed; added code to set tdmrdone=.TRUE. and call TDMR(2) just before TRAC-M exit if current calculation is coupled to kinetics (itdmr=1)

TracInputM.f90 (input) - added NAMELIST variables itdmr and itmrp; added code for reading in and performing error checks on these flags
trans.f90 - See steady.f90

VessTF3DSM.f90 (tf3ds) - activated coding for depositing power into the coolant of VESSEL components