PU/NE-98-30

MODIFICATIONS TO PARCS/PDMR TO ACCOMMODATE RESTART AND STEADY-STATE INITIALIZATION IN THE COUPLED CODE

Douglas A. Barber, 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

Contract NRC-04-97-046, Task 8
Performed Under Subcontract to SCIENTECH, Inc.

November 1998

I. Introduction

This document describes the requirements and design modifications necessary to provide a steady-state initialization and full restart capability within RELAP5/PARCS. These capabilities are needed to provide the user with the flexibility to run RELAP5/PARCS for a wide range of applications. Specifically, the steady-state initialization is necessary for Hot Full Power (HFP) transients in order to obtain equilibrium between the thermal-hydraulic/neutronic field equations prior to running a transient application. The restart is needed for, but not limited to, transitioning between steady-state and transient calculations. The following two sections discuss in detail the changes needed to both PARCS and the PARCS-Specific Data Map Routine (PDMR). Section III then provides verification of the steady-state and restart capabilities.

II. Software Requirements Modifications

The current version of PARCS (NRC-V1.00) has both a steady-state and a restart capability. However, these options were implemented for the stand-alone execution only. The following sections describe the PARCS/PDMR requirements necessary to implement a steady-state and full restart capability for the coupled RELAP5/PARCS code. These requirements supplement those described in the Software Requirements Specification (SRS) for the PDMR[1], and the PARCS Manual[2].

II.A. Restart Capability

The implementation of a restart capability within the framework of the coupled RELAP5/PARCS code will provide the user with the following options:

(1)	restart	steady-state	(from a steady-state calculation)
(2)	restart	transient	(from a steady-state calculation)
(3)	restart	transient	(from a transient calculation)

As a fourth option, the ability to initially run RELAP5 without PARCS (e.g. use a power table, control variable, or point kinetics), and then restart with the PARCS option activated, will also be available. The logic needed to treat restarts already exists in the PDMR, and only the logic paths in the PARCS source need to be modified.

II.B. Steady-State Initialization Capability

The ability to provide a steady-state initialization treatment in the coupled RELAP5/PARCS code requires very few modifications to the existing source. Specifically, the PARCS source would remain largely unchanged, and the PDMR source would have to be modified only to

receive the indication from the RELAP5-Specific Data Map Routine (RDMR) that RELAP5 is performing a steady-state calculation. This could easily be accomplished by creating an additional logical address in the existing initial T/H control buffer. It should be noted that in order to maintain coherency between the PDMR and the RDMR, this logical needs to exist in the initial control buffer sent by the RDMR.

In addition, a steady-state initialization could also be performed by running RELAP5 in transient mode without external perturbations. In this situation, the user input for PARCS could be changed to specify that PARCS should be run in steady-state mode even though RELAP5 is running in transient mode.

III. Design Modifications

The software design changes necessary to implement the requirements specified in Section II are described in detail for both the PARCS and the PDMR source in the following sections.

III.A. PARCS Source Modification

As was mentioned, the current version of PARCS has a restart capability only for the stand-alone execution. Thus, the modifications to the PARCS source deal with implementing the logic necessary to provide this restart capability within the framework of the coupled RELAP5/PARCS code. In addition, modifications to the logic path in the steady-state calculation had to be made in order to provide an initialization capability for RELAP5/PARCS. The following subsections describe the necessary source changes.

III.A.1. parcs.f

The changes to this routine to handle the steady-state initialization were in fact very minor. As was mentioned in Section II.B, there are two ways that PARCS could engage its steady-state initialization logic. First, if RELAP5 sends a signal indicating that it is executing in steady-state mode, and second, if RELAP5 sends a transient execution signal, but PARCS input indicates that the steady-state logic should be utilized. The determination of steady-state is demonstrated in the following:

```
if (extth) call PDMR(1)
if (extth .and. .not.ssinit .and. .not.tran) ssinit=.true.
if (ssinit) tran=.false.
```

In subroutine `PDMR(1)`, the PDMR has received a calculation mode signal (e.g. steady-state or transient) from RELAP5, and has stored this indication in the PARCS logical, `ssinit`. If `ssinit` is “true”, indicating that RELAP5 is running in steady-state, then PARCS will also exe-

cute its steady-state logic. However, if `ssinit` is returned from `PDMR(1)` with a value of "false", PARCS checks to see if the user specified a steady-state calculation in the PARCS input (i.e. `.not.tran`). If this is the case, `ssinit` is set to "true". In either case, once `ssinit` has been set to "true", PARCS steady-state eigenvalue is placed in a time-dependent loop which is terminated upon indication from RELAP5 that the calculation is complete. This logic is shown below:

```
 100  continue
      if (extth) call PDMR(2)

         . . . .  < steady-state eigenvalue calculation >

 200  if (extth) call PDMR(3)
      if (ssinit .and. .not.calcterm) goto 100
```

Concerning the restart capability, if the steady-state initialization logic is being executed in PARCS, then `parcs.f` is responsible for calling the subroutine which dumps steady-state restart data. This determination is made following the call to the second functional unit of the PDMR, and is based on the restart edit signal (written to PARCS variable `rsted` in `PDMR(2)`) which is sent by the RDMR:

```
      if (extth) call PDMR(2)

         . . . .

      if (extth .and. ssinit .and. rsted) then
        irstadv=irstadv+1
        call rstedit(-1)
      endif
```

In the above logic, the restart data is written at the beginning of the next time step using the data from the end of the previous time step, which is consistent with RELAP5's restart editing. A restart block counter, `irstadv`, is incremented to tag the block of data being written. This block number will be used for restart calculations to determine which block of data should be read.

When a restart calculation is being performed, it is necessary to determine whether the current calculation is being restart from a steady-state or a transient calculation. This determination is made by calling subroutine `restart.f`, described in Section III.A.4. If the calculation is being restarted from a steady-state calculation, then `restart.f` will call the subroutine for processing the restart input file. If the current calculation is then another steady-state initialization, then `restart.f` is called again to open the restart output file. This logic, which follows the call to the initialization unit of the PDMR, is shown below, where the subroutine argument in the call to `restart.f` is used to indicate which functional unit should be executed.

```
c open input restart file
      call restart(0)
c open output restart file if performing ssinit
      if (ssinit) call restart(1)
```

If the current execution is a transient calculation which is being restarted from a steady-state calculation, then `parcs.f` will execute the steady-state logic once at the beginning of the calculation to determine the core k-effective and initial power distribution, and then call the transient subroutine. However, if the current execution is a transient calculation which is being restarted from a transient calculation, then `parcs.f` will bypass the initial steady-state step, and proceed directly to the transient calculation, as shown below:

```
      if (extth .and. rstrt .and. .not.ssdata) then
        call init
        goto 105
      endif

         . . . .  <steady-state logic>

 105  if(tran) call transient
```

III.A.2. inittran.f

The modifications to inittran.f relate only to implementing the restart capability for transient cases. As was mentioned in the previous section, input restart data is processed at the beginning of the PARCS execution if the calculation is being restarted from a steady-state calculation. However, if the current execution is being restart from a transient calculation, it is necessary to delay reading the restart data until the beginning of the transient calculation. This transient restart data is processed at the beginning of subroutine `inittran.f`, as shown below:

```
c process transient restart data
      if (extth) then
        if (rstrt .and. .not.ssdata) call rstinp
      else
         . . . .
      endif
```

In addition, if the current execution is being restart from a transient calculation, it is necessary to bypass the initializing of decay heat parameters, delayed neutron precursor concentrations, and steady-state feedback variables, since this data is stored in the restart file. The following demonstrates this logic:

```
      if (rstrt .and. .not.ssdata) goto 100

     . . . .  < initialize decay heat parameters for transient >

 100  continue

         . . . .
```

```
      if (rstrt .and. .not.ssdata) goto 110

     . . . .  < calculate steady state precursor concentration >
     . . . .  < save initial feedback variables >

 110  continue
```

Prior to exiting `inittran.f`, the output restart file is opened through the call to subroutine `restart.f`, and the steady-state data read from the input restart file is dumped back to the new output restart file, as demonstrated below:

```
      if (extth) then
        call restart(1)
        call rstedit(0)
        if (.not.rstrt) irstadv=-1
      else
         . . . .
      endif
```

III.A.3. transient.f

The modifications to `transient.f` relate only to incorporating the restart logic. First, upon receipt of the restart edit signal (`rsted`) sent from the RDMR, `transient.f` calls the subroutine which dumps transient restart data, as shown below:

```
           if (rsted) then
             irstadv=irstadv+1
             call rstedit(1)
           endif
```

In the above logic, the restart data is written at the beginning of the next time step using the data from the end of the previous time step, which is consistent with RELAP5's restart editing. A restart block counter, `irstadv`, is incremented to tag the block of data being written. This block number will be used for restart calculations to determine which block of data should be read.

The second change is needed for cases where the current execution has been restarted from a transient calculation. Specifically, an initial "stutter" step is used such that the transient calculation logic is bypassed, and the initial restart power distribution read from the restart file is sent to RELAP5. The following demonstrates the necessary logic:

```
        if (rstrt .and. .not.ssdata) then
c if restarting from transient, the first step is a "stutter" to
c send initial power to T/H process.
          skip=.true.
        else
         . . . .
        endif

         . . . .
```

```
        do while (.not. calcterm)
           call PDMR(2)
           if (skip) then
             skip=.false.
             call xsecfb
             call relpow
             if (popt(5)) then
               call calrho(rhoadj)
             endif
             goto 888
           endif

         . . . .  < transient calculation >

  888      continue
           call PDMR(3)
        enddo
```

III.A.4. restart.f

A new subroutine, `restart.f`, was added which control all necessary I/O for the input and output restart files. This subroutine has three functional units, where the entry into these units is determined by a subroutine argument, `iopen`. If `iopen` is equal to zero, then restart.f inquires in the existence of the input restart file. If this file exists, it is opened, and the first line of data is processed. This first line indicates whether the data written in the restart file corresponds to a steady-state (`ssdata`="true") or a transient (`ssdata`="false") calculation. If `ssdata` is equal to "true", then a call is made to the subroutine which controls the steady-state restart input processing. The following demonstrates this logic:

```
      if (iopen.eq.0) then
        if (rstrt) then
          inquire(file=filename(irstin),exist=filexist)
          if (.not.filexist) then
            write(ioutp,*)" !! Restart File Does Not Exist !!"
            if (extth) then
              errparcs=.true.
            else
              stop
            endif
          else
            open(irstin,file=filename(irstin),status='unknown'
     &,           form='unformatted')

c determine whether restart data was written during Steady-State
c (ssdata=true) or Transient (ssdata=false) execution.
            read(irstin) ssdata
            if (ssdata) call rstinp
          endif
        endif
```

If `iopen` is equal to one, the output restart file is opened and the logical `ssinit` is written to the first line. As discussed above, this logical will be used upon restart to determine what type of data was written to the restart file. This source is shown below:

```
      elseif (iopen.eq.1) then
        open(irstout,file=filename(irstout),status='unknown'
     &,      form='unformatted')

c write ssinit variable... this variable will be processed to
c determine whether restart data was written during Steady-State
c or Transient execution.
        write(irstout) ssinit
```

Finally, if `iopen` is equal to two, then the input and output restart files are closed:

```
      elseif (iopen.eq.2) then
        close(irstin)
        close(irstout)
      endif
```

III.B. PDMR Source Modification

The modifications to the PDMR source deal only with implementing the logic necessary to provide a restart and steady-state initialization capability within the framework of the coupled RELAP5/PARCS code. These modifications were applied to the logic documented in the Software Design and Implementation Document (SDID) for the PDMR[3]. Only one source file, `p-initc.f90`, needed modification, which is discussed below.

The first change to `p-initc.f90` relates to the steady-state initialization capability of the coupled RELAP5/PARCS code. Specifically, the size of the initial T/H logical buffer was increased by one, where this additional address was used to receive the indication of a steady-state or transient run from the RDMR. This indication is stored in PARCS logical, `ssinit`. The second change simply involved commenting out the assignment of PARCS variable, `rstrt`. This was done because PARCS now determines whether to run a restart case based on its own user input. The following demonstrates these changes:

Original Source:

```
c Extract data from RELAP5 control buffers.

         . . . . .

      done = lbufth(6)
      rstrt = lbufth(8)
      time = r8bufth(1)
```

New Source:

```
c Extract data from RELAP5 control buffers.

         . . . . .

      done = lbufth(6)
c      rstrt = lbufth(8)
      ssinit = lbufth(9)
      time = r8bufth(1)
```

IV. Verification

The test model utilized to verify the steady-state initialization and restart capabilities is based on the RELAP5 Typical PWR model and a PARCS NEACRP PWR core model, as described in the initial completion report for the coupled RELAP5/PARCS code[4]. The specific test cases which were run are listed below:

(1) New steady-state for 1.0 second
(2) Steady-state restarted from (1) at 0.5 seconds; run to 1.0 second
(3) Transient restarted from (2) at 1.0 second; run to 2.0 seconds
(4) Transient restarted from (3) at 1.5 seconds; run to 2.0 seconds
(5) RELAP5 new steady-state run w/o PARCS for 1.0 second;
Restart transient run with PARCS

It should be noted that for the steady-state runs the transient trip cards were turned off. The results of these test cases are described in the following sections, and are consistent with those shown in the RELAP5/RDMR modification document for steady-state and restart capabilities[5].

IV.A. Results of Case 1

The core k-effective from the initial steady-state case is shown in Figure 1. This result will be used to determine the relative error in the steady-state restart result of Case 2.

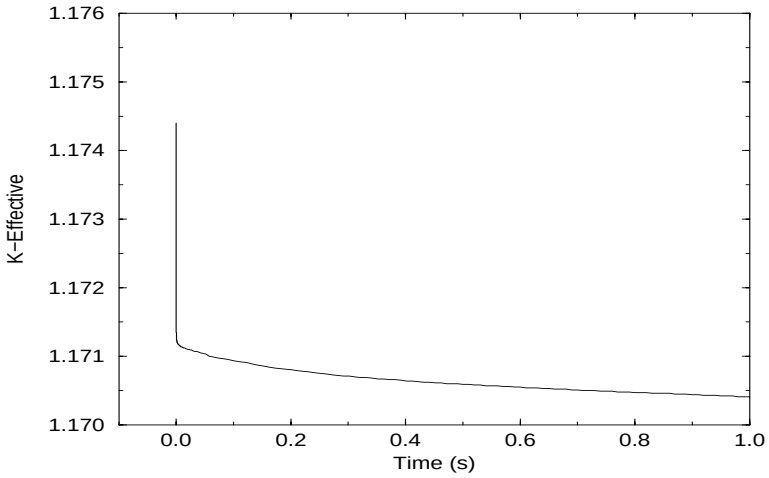

Figure 1: Steady-State Result for Case 1

IV.B. Results of Case 2

The relative error in core k-effective for the restarted steady-state case is shown in Figure 2. The non-zero relative error is minor and can be neglected.

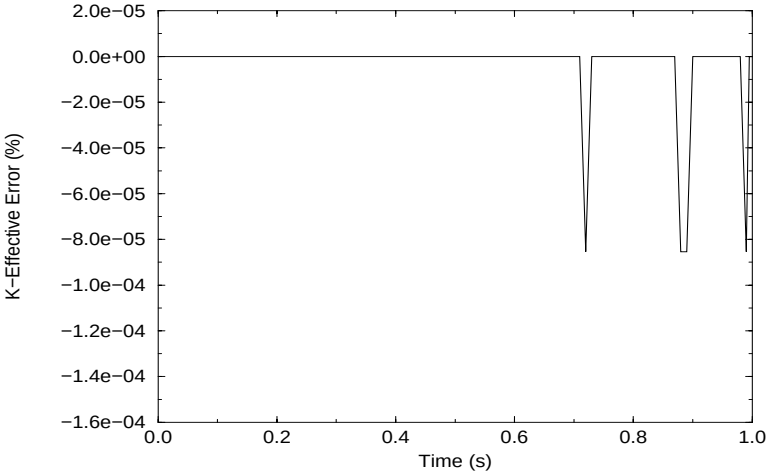

Figure 2: Steady-State Restart Relative Error

IV.C. Results of Case 3

The restarted transient result is shown in Figure 3. The core reactivity and relative power response corresponds to a small break LOCA which was initiated at time zero by RELAP5. This result will be used to determine the relative error in the transient restart result of Case 4.

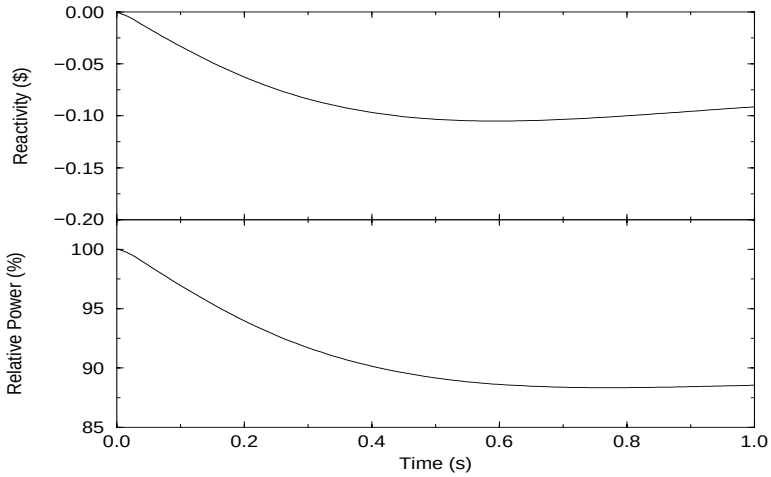

Figure 3: Transient Restart from Steady-State Result

IV.D. Results of Case 4

The relative error in core reactivity and relative power for the restarted transient case is shown in Figure 4. As can be seen, the transient restart produced no differences compared to the original transient run.

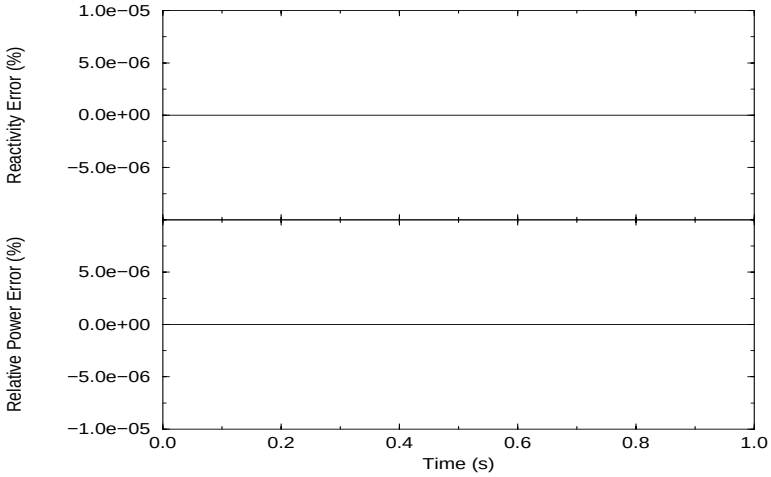

Figure 4: Transient Restart Relative Error

IV.E. Results of Case 5

The transient core reactivity and relative power response for Case 5 is shown in Figure 5. This transient case involved restarting RELAP5 from a RELAP5/Point Kinetics steady-state case, and running PARCS in transient mode. Thus, the difference in this result compared to that shown in Figure 3 is due to the fact that the power shape computed by PARCS during the restarted transient is inconsistent with that used in the point kinetics model during the steady-state. Nonetheless, this results verifies the functionality of the fourth restart option described listed in Section II.A.

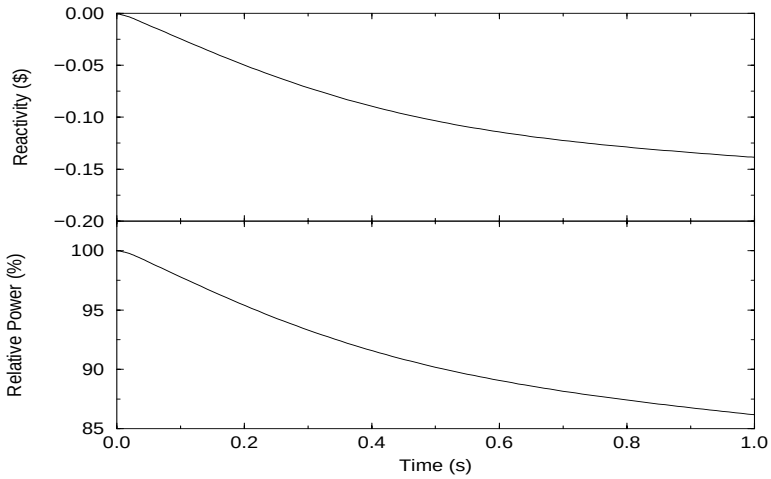

Figure 5: Transient Result for Case 5

V. Summary

The implementation of a steady-state initialization and full restart capability has been presented and verified in the previous sections. The changes which were necessary to the PARCS and PDMR source were minimal, but they provide the flexibility to run RELAP5/PARCS for a wide range of cases. This document should serve as an addendum to the SRS[1] and SDID[3] for the RDMR.

VI. References

1. D. Barber and T. Downar, "Software Requirements Specification for the PARCS-Specific Data Map Routine in the Coupled RELAP5/PARCS Code," Technical Report, PU/NE-98-11, SCIENTECH, Inc., (1998).

2. H. Joo, D. Barber, G. Jiang, and T. Downar, “PARCS: A Multi-Dimensional Two-Group Reactor Kinetics Code Based on the Nonlinear Analytic Nodal Method; NRC-V1.00,” Technical Report, PU/NE-98-26, Purdue University, (1998).

3. D. Barber and T. Downar, "Software Design and Implementation Document for the PARCS-Specific Data Map Routine in the Coupled RELAP5/PARCS Code," Technical Report, PU/NE-98-13, Purdue University, (1998).

4. D. Barber, T. Downar, W. Wang, and G. Mortensen, "Completion Report for the Coupled RELAP5/PARCS Code," Technical Report, PU/NE-98-18, Purdue University, (1998).

5. D. Barber, T. Downar, and W. Wang, “Modifications to RELAP5/RDMR to Accommodate Restart and Steady-State Initialization in the Coupled Code,” Technical Report, PU/NE-98-29, Purdue University, (1998).