**Proceedings of the ASME 2010 3rd Joint US-European Fluids Engineering Summer Meeting and 8th International Conference on Nanochannels, Microchannels, and Minichannels**
**FEDSM-ICNMM2010**
**August 1-5, 2010, Montreal, Canada**

**FEDSM-ICNMM2010-' 0( $%**

# REFORMULATION RELAP5-3D IN FORTRAN 95 AND RESULTS

**George L Mesina**
Idaho National Laboratory
P.O. Box 1625
Idaho Falls, ID, USA 83415
Email: George.Mesina@inl.gov

## ABSTRACT

RELAP5-3D is a nuclear power plant code used worldwide for safety analysis, design, and operator training. In keeping with ongoing developments in the computing industry, we have re-architected the code in the FORTRAN 95 language [2], the current, fully-available, ANSI standard FORTRAN language. These changes include a complete reworking of the database and conversion of the source code to take advantage of new constructs. The improvements and impacts to the code are manifold. It is a completely machine-independent code that produces machine independent fluid property and plot files and expands to the exact size needed to accommodate the user's input. Runtime is generally better for larger input models, many prior user-reported problems have been resolved, and the program is better tested. Other impacts of code reformulation are improved code readability, reduced maintenance and development time, increased adaptability to new computing platforms, and increased code longevity. Comparison between the pre- and post-conversion code are made on the basis of programming metrics and code performance.

## INTRODUCTION

RELAP5-3D solves multi-dimensional, multi-phase mass, momentum, and energy equations, multidimensional heat transfer equations, and multi-dimensional neutron kinetics equations. The code implements trips, controls, and physical models specific to power plants. With these physical models, RELAP5-3D has been applied to multiple types of nuclear reactors, such as Pressurized Water Reactors (PWR), Boiling Water Reactors (BWR), Liquid Metal Fast Reactors (LMFR), and the Next Generation Nuclear Plant (NGNP). It has also been used to model fusion reactors and steam supply systems.

Many large physics codes, such as RELAP5-3D, have been developed over a period of years, and numerous large and multi-purpose codes are under development today. All codes which become successful will develop a user community and a virtual library of input files distributed across the computing world. The longer the code is successful, the more development it will undergo, expanding to meet the ever-changing needs of its user community. This is generally accompanied by growth of its user community, libraries and the overall investment made in the large code.

Over the same period of time, the face of computing will re-invent itself. The large code will have had to adapt to fundamental changes in the computing industry including operating systems, software support libraries, computing paradigms (such as parallelism) either merely to continue to perform or to meet the needs and desires of its user community. After a sufficient number of adaptations and expansions, codes often lack cohesion due to the number and variety of developments, developer's styles, error corrections and software patches. Some subprograms may have grown to unwieldy size or have expanded function well beyond original design specifications. Inefficiencies will often have resulted, both in execution and programming. There may be unused code and even entire subprograms and procedures no longer in use. Documentation often no longer corresponds to source code. In short, the program needs to be reworked.

A conversion from one language to another, or even from one language level to another, is often the impetus for reformulating a successful program. A direct translation from language to language will result in a program with the same lack of cohesion and worse than that, as the constructs of the previous language or level may have been eliminated or modified slightly. Thus the resulting code has new sources of inefficiency and error, namely those due to translation.

 Copyright © 2010 by ASME

Therefore, the program should be reformulated for the new language or language level. This requires a reworking of the database so that it is easy to develop with and efficient within the concepts, computing paradigms, and coding constructs of the new language. The source code should be translated, re-factored, or rewritten to take advantage of the new language capabilities and features, the paradigms of computing, and to be able to grow easily with future developments in the computing industry. Portability, legibility, maintainability, ease of development, and longevity are key elements to consider with redesigning a code and its database.

Some examples of reformulated programs include TRACE [3], LPCIS [4], and RELAP5-3D [5]. The reformulation of RELAP5-3D is reported here. The strategy and method for performing this rework of the code were rather generic in nature; they could be repeated, with some modification, for many other large and successful codes. However, the presentation of the strategy and method here is restricted to the reformulation of RELAP5-3D.

Section 2 explains the conversion methodology at a high level. Section 3 covers the reformulation of the database. Section 4 presents the conversion of the source code from FORTRAN 77 to FORTRAN 95. Section 5 covers the testing metric and methodology. The results are in Section 6; it provides analysis of the new code, by both static and dynamic measures, and summarizes improvements for code users.

## NOMENCLATURE

| | |
|---|---|
| ANSI | American National Standards Institute |
| BWR | Boiling Water Reactors |
| COBRA | Coolant Boiling in Rod Arrays |
| CPU | Central Processing Unit |
| FA | Fast Array |
| FORTRAN | FORMula TRANslation |
| HSG | Heat Structure Geometries |
| IA | Integer fast Array |
| LMFR | Liquid Metal Fast Reactors |
| NGNP | Next Generation Nuclear Plant |
| PVM | Parallel Virtual Machine |
| PWR | Pressurized Water Reactors |
| RELAP5 | Reactor Excursion and Leak Analysis Program |
| RGUI | RELAP5 Graphical User Interface |
| SCDAP | Severe Core Damage Analysis Package |
| UP | User Problems |
| XDR | eXtended Data Representation |

## CONVERSION METHODOLOGY

The conversion was broken into subtasks that consisted of designing and transforming the database, converting the coding related to the database, and testing in stages. Prior to the subtasks, preliminary tasks that simplify the transformation were completed. Figure 1 gives a high-level depiction of the overall process.

Both commercial and internally developed software were applied to carry out the transformation, conversion, and testing. The use of computational software to precisely perform repetitive operations reduced both time and manual errors; it allowed recognition and solution of conversion problems to take place at a high level.

A number of preliminary modifications were performed prior to transforming the database; see Figure 1. Unused data and coding were identified and eliminated. For the source code, unused and obsolete code was removed, combinations of multiple bit-operators with powers of two were replaced by the FORTRAN 95 intrinsic functions, and source code was reformulated in the structured programming paradigm [6]. Details of the preliminaries are given elsewhere [6, 7].

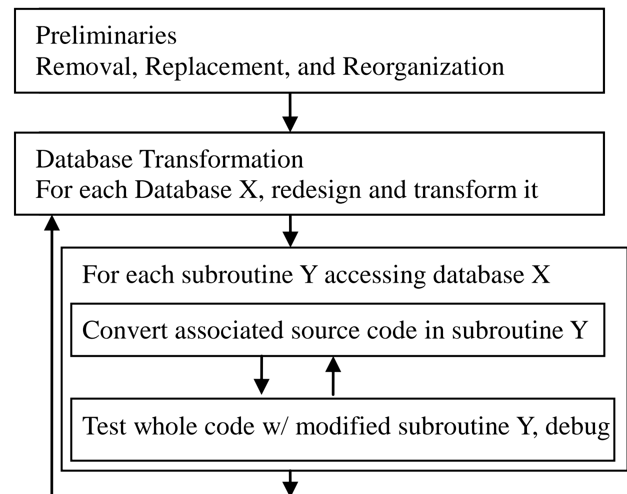


**Figure 1. Conversion Strategy**

As seen in Figure 1, a database is selected, redesigned, and transformed first, then all the source code associated with it is converted to access it. This is done for one subprogram, then the whole program is then tested with the modified subprogram on a suite of input models to verify that the calculations do not change. If answers do change, debugging ensues. This process is repeated until there are no further accesses to the old form of the database. During the reformulation task, other development tasks proceeded simultaneously that seriously complicated it. Thus this process is designed for flexibility and precision.

Database transformation, source code conversion, and testing are covered in the subsequent sections.

## DATABASE TRANSFORMATION

Prior to conversion, the memory-saving databases RELAP5-3D Version 2.4 was comprised of 47 major databases, called internal files. The first step was to write a database conversion program and select a simple database to convert; this helped test and debug the conversion software. It was then applied to successively more complicated conversion tasks until obstacles to conversion were encountered. Solutions were developed and the software was continually upgraded to


Copyright © 2010 by ASME

correctly convert more complex databases and subprograms. Some language constructs could only be transformed manually. A few databases and subprograms were manually rewritten in FORTRAN 95 due to their complexity.

In algorithmic form, the procedure for transforming the database can be expressed as follows:

*Database Transformation Algorithm*

1. Analyze the database as a whole to devise means to subdivide into smaller subtasks.
2. Order the transformation subtasks.
3. Analyze the (first) next subtask's database and redesign it (if necessary).
4. Implement the new design in FORTRAN coding.
5. If any subtasks remain, return to step 3

Many large computer codes, such as RETRAN-03 [8], Cobra [9], Lava [10], NESTLE [11], Trac-PF1 [12], and RELAP5 store virtually all data in a one-dimensional container array. This was done to avoid computer charges for memory usage on mainframe computers until the early 1990s. With data as linear arrays within a linear container, all data could be easily shifted to the container's lowest indices after input processing and the unused portion returned to the operating system. The container is called FA (Floating-point Arrays) and IA (Integer Arrays); the two are made equivalent via the equivalence statement. This storage system has become unnecessary because modern computing platforms do not charge for memory.

There are numerous problems with this memory-saving database. First, the linear container is unnatural for representing character and logical data. Second, it requires at least two extra lines of code per loop. It has coding tricks and pitfalls that do not occur in a more standard data arrangement. Finally, the database design requires that integer, floating point, logical, and character data be included in a single array, and be made equivalent to each other in violation of the ANSI FORTRAN 77 Standard.

Memory is subdivided in the FA array into internal files, one for each of the 47 major databases. These databases contain information about related quantities such as control systems, trips, general tables, heat structures, control volumes, neutron kinetics, etc. Some databases are further subdivided, e.g. point and multidimensional kinetics, or variable and logical trips. Some databases exist to relate two other databases efficiently to reduce run time, such as the inverted junction table and list vectors. Most databases, or files, exist only if the model requires them. Some exist only for input processing and are eliminated thereafter. Each database is assigned a "file" number. For example, control volume database is file 4, radionuclide transport data is file 47, etc. Array FILNDX points to the beginning of each internal file in the container; e.g. FILNDX(4) = K means the first control volume datum is at IA(K).

The subtasks are to transform the major databases into a new form while possibly combining, splitting, or eliminating some. The order for processing these databases was established by their internal complexity, which ranged from simple to complex. Simple databases consist of arrays of uniform length. Slightly complex database have two or more sets of uniform length arrays, where the array length of the sets are different. Moderately complex databases contain arrays of non-uniform length. Hierarchically complex databases have two levels of data where an element of the top level has a complete set of lower level data associated with it. The most complex databases contained multiple levels of fixed and varying length arrays. The transformation order was from simplest to most complex with database size breaking ties.

Complexity was caused by data layout. The arrays of a database were positioned with the first entry of each array arranged consecutively in memory, then all the second entries, etc. See Figure 2. Equivalence statements specified *relative*, not actual, indices of arrays within the container array. Figure 2, shows arrays A, B, and C of fictitious file 48 as they align with FA/IA in memory along with their declaration statements. For file 48, FILNDX(48) = 80 and the skip factor is 3; thus A(2) = FA(81+3). In simple database L with M arrays of fixed length N, IA(FILNDX(L)) = N, and arrays begin at FILNDX(L)+1. The formula to access entry J of C, its 3rd array, is C(FILNDX(L) + (J-1)*M). In a loop, I is initialized to FILNDX(L) and incremented by M at loop bottom, then C(I) = C((J-1)*M + FILNDX(L)).

```
A(1),    B(1),    C(1),    A(2),    B(2), . . .
FA(81), IA(82), FA(83), FA(84), IA(85)
```

```
integer B(1)
real*8 A(1), C(1)
equivalence (FA(1),A(1)), (IA(2),B(1), (FA(3),C(1))
```

**Figure 2. Original Data Layout and Declaration Example**

In step three of the transformation procedure, the major databases are transformed into improved forms for use with FORTRAN 95 coding. *Simple databases are transformed so that their arrays are contiguous*. In step 4, a FORTRAN 95 module is created that declares the arrays, has array length as a scalar, and there is no skip factor. The module also documents all its variables and has internal subroutines that operate on the data it declares.

*Databases with greater complexity require greater amounts of redesigning*. Consider the heat transfer file, a hierarchical database of 2 levels and varying length arrays. Its data represents the flow of heat through solids, such as pipe walls, and to the fluid. Data is organized into two-dimensional grids of points, called Heat Structure Geometries (HSG), with one boundary on the inside of a wall (or solid's centerline) and the other on its outside. Temperatures are measured at grid points. A model may have many HSG. each with a different numbers of rows, called heat structures, and columns, called mesh lines,


Copyright © 2010 by ASME

across which the material could change, such as from fuel to cladding represented as materials A and B in Figure 3a.

The heat database is implemented in comdeck HTSRCM which has K HSG where the $i^{th}$ HSG has $N_i$ heat structures, and $N = N_1 + . . . + N_K$. Its 5-part layout is pictured in Figure 3b. The scalar N is the first part. The second part is an index array to the start index in FA of each HS. The data for the heat structures are stored in the third portion, the HSG geometry, material and weighting data came fourth. The old and new temperatures are stored in the fifth section. Each HS has an index (pointer) to its temperatures and HSG. HSG data has an index to its first HS. Data for the linear system that represents the heat conduction equation is stored separately in temporary (or scratch) database, HTSCR.

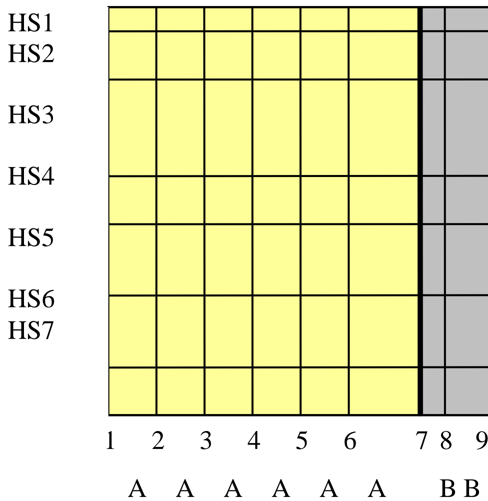


**Figure 3a. A Heat Structure Geometry**

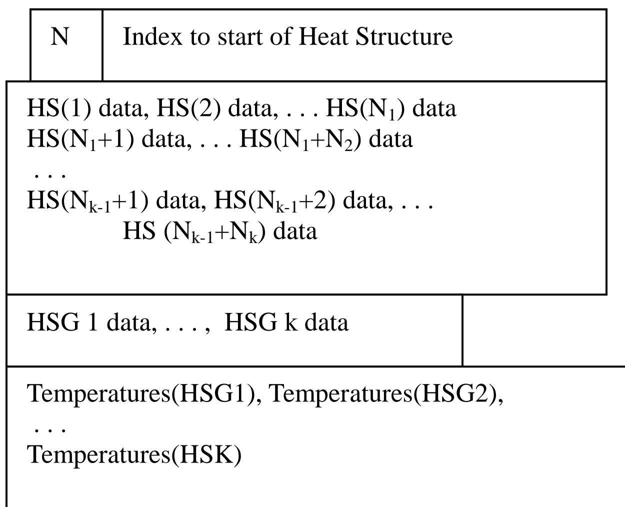


**Figure 3b. Heat Transfer Data Layout**

The redesign of the heat structure database is shown in Figure 4. The solution data for the heat conduction equations have been incorporated. The new form is implemented in derived types.

To see the improvement, consider accessing the old-time temperature at point k of the $j^{th}$ heat structure of HSG k in both databases. In the memory-saving database, FILNDX(8)+1 is the location in the IA array of the unnamed index array to HS.

- Set INDEX1 = the IA-index of first HS of HSG k in the list of all heat structures.
- Set INDEX2 = IA(FILNDX(8)+INDEX1+j-1) is the index of the first element of HS(j).
- Set INDEX3 = TMPNDX(FILNDX(8)+INDEX2+k-1) is the temperature.
- TMPO(INDEX3) = the temperature from previous time advancement.

FILNDX(8) is added to INDEX1 and INDEX2 because “files” store offsets from the file’s starting index only.

In the *new database*, the temperature at previous time in HSG i at grid point (j, k), is simply:

- htg(i)%temp(2,j,k),

where htg is the HSG derived type array for accessing temperatures. The htg(i)%temp array stores two temperatures at each grid point, differentiated by the first of its three subscripts; the first subscript is one for new time temperature and 2 for old time temperature.

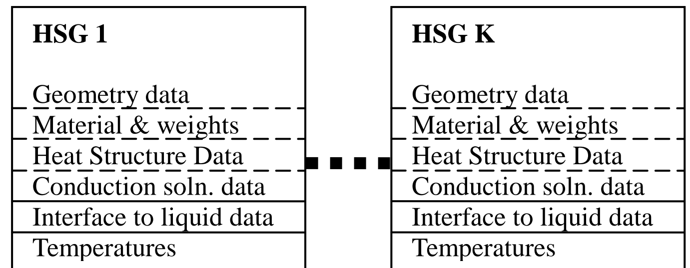


**Figure 4. Module Memory Layout for Heat Transfer**

Step 4 of the database transformation algorithm employed a program written for the purpose. Its input is the comdecks prescribing the database; its output is a FORTRAN 95 module.

The major databases were split into a data dictionary comdeck and one or more comdecks of declaration and equivalence statements. Often these had a variety of unrelated names that were unhelpful for locating and accessing data. The new database design allows exactly one module for each database with a uniform naming convention for easy recognition; all module names end with the three letters “mod” while the first 3-6 letters describe its data. Internally, all modules have a standard three-part design: declarations, data dictionary, and internal subprograms. Within part 1, data is declared in four sub-parts: created derived types, derived type arrays, standard-type arrays, and scalars. All variables are listed in alphabetical order within these four sub-parts sorted in


Copyright © 2010 by ASME

alphabetical order. Part three has four common internal subroutines: allocatable array constructor and destructor routines and restart read and write routines. Some modules have additional routines for such purposes as initializing, counting, and data transferring.

The transformation program was written to reduce conversion time and to reduce or eliminate manual conversion errors. It analyzed both type and equivalence statements, sorted the arrays into alphabetical order, and created alphabetized, size-and-type declaration statements that were placed within a derived type declaration. The program output was a module with identification comments, all declarations of part one, including the allocatable derived type array, part two built from the data dictionary comdeck, and part three with generated internal subroutines for allocation, deallocation, and data transfer. The program was first applied to the comdecks of simple databases, then improved for more complex ones. Manual modifications of its output were necessary for databases being reworked, such as the heat data.

Of the 47 databases, 36 were converted, 6 were eliminated, 2 were postponed, and 3 were subsumed into other databases. By postponing conversion, of the RELAP5 Graphical User Interface (RGUI) and Severe Core Damage Analysis Package (SCDAP) the features have become temporarily unusable

## SOURCE CODE TRANSFORMATION

Transforming source code from one language or level of language to another is a difficult task for a large computer program. Transforming source code while the program is under active development and in heavy use further complicates the task. The program must work at every version for code users. The strategy devised and implemented for conversion worked satisfactorily.

The strategy was to convert the code database by database. First the database was transformed, then each subroutine that used it was modified in turn. Every variable belonging to the database was changed from its form in the original database to its form in the new database. Testing was performed after every group of subroutines to ensure that all test cases were run correctly; this is covered in Section 5.

Some program constructs and programming techniques that were part of older FORTRAN language levels were obsolescent, incompatible, unneeded, or even problematical in FORTRAN 95. For example, memory allocation and pointers were not part of FORTRAN predecessors of the 1990 standard. These features were replaced, sometimes to the great improvement of the source code. The result is a program with constructs and modern programming techniques of standard ANSI FORTRAN 95. A brief description of the reworking of the language is presented in this section.

Due to equivalencing of IA and FA and the need to support both 32- and 64-bit integer platforms, a preprocessor was used. It would convert integer arrays of the IA container into tiny length-two arrays. That was needed so that indexing of equivalent integer and floating point quantities would be identical. This created numerous issues, particularly with subprogram call arguments and debugging.

By eliminating the container array, this confusing coding practice has been eliminated. All extra overhead incurred through this database, such as FILNDX, skip-factors, and offsets within databases and portions of databases are eliminated. In particular, pointers have been immensely simplified. Previously, these were implemented via indexing within the container array, and this required the use of tables of indices, offsets to the tables, offsets to the target database, and offsets within the target database. This has been replaced by true FORTRAN 95 pointers. The overhead of pointer offsets and indices has been eliminated.

The use of integers to hold character data in the container array, a carry-over from FORTRAN 66 which had no character data type, was eliminated. It was necessary to use internal reads and writes to move data for various purposes. The programming, readability, and debugging issues that this engendered have been removed by converting the character containing integers to true character variables.

Another readability issue was variable reuse. With larger computer memories, it is unnecessary to reuse variables to save memory. Most every array is given its own identity, even scratch arrays. This removes a large source of errors.

An illustration of source code transformation is given in Figures 5a and 5b. Actual subroutines of RELAP5 are either too small to illustrate much transformation or too large to include in full. These Figures represent some of the relevant declarations and coding from a portion of the trip subroutine that processes logical trips.

In Figure 5a, note the “n = filndx(18)” statement and, just above the “end do,” the statement “n=n+ntlskp.” The transformation eliminates these in Figure 5b. The “include” statements are replaced by “use” statements. Another use of pointers, and a small change to input processing, reduces the calculation of ltrip and rtrip from twelve lines to two in Section 2.1. The obsolescent “computed go to” is replaced by a case statement in Section 2.2. The logic in the if-test following statement label 63 in Figure 5a is simplified in 5b in Section 2.3. The declaration of real pointer “triptime” reduces five statements to one in Section 2.5.

Several changes were made purely for readability. Four-character comparative operators, E.G. “.ge.” are replaced by one-two symbol operators such as “>=.” Use of the “iand” intrinsic has been replaced by “btest” and “ibits” intrinsics. Many comments have been introduced with outline-style numbering.

The restructuring performed during the preliminaries phase is responsible for the elimination of the computed go to and the uniform indentation.

Overall, despite the introduction of documentation, the transformed source code is shorter.


Copyright © 2010 by ASME

```
      subroutine trip
c  Tests trip conditions and sets trip conditions and time of trip.
#include "comctl.H"
#include "contrl.H"
#include "fast.H"
#include "trpblk.H"
      logical ltrip,rtrip
c
      n = filndx(18)
      timx = timehy
      timehy = timehy - dt
c
c  Logical trips.
      if (ntrpnl(filndx(18)) .eq. 0) go to 100
      do i = 1,ntrpnl(filndx(18))
        if (iroute .ne. 1 .or.
     &      (iroute .eq. 1 .and. iand(imdctl(1),64).eq.0)) then
          ltrip = trptm(ntrtr1(n+1)).ge.0.0 .neqv. iand(ntrpop(n),16)
     &    .ne.0
          rtrip = trptm(ntrtr2(n+1)).ge.0.0 .neqv. iand(ntrpop(n),32)
     &    .ne.0
        else
          ltrip = trptmss(ntrtr1(n+1)).ge.0.0 .neqv. iand(ntrpop(n),16)
     &    .ne.0
          rtrip = trptmss(ntrtr2(n+1)).ge.0.0 .neqv. iand(ntrpop(n),32)
     &    .ne.0
        endif
        m = ishft(ntrpop(n),-24)
        if (m - 2) 71,72,73
   71   ltrip = ltrip .or. rtrip
        go to 63
   72   ltrip = ltrip .and. rtrip
        go to 63
   73   ltrip = ltrip .neqv. rtrip
c
   63   continue
        if (iroute .ne. 1 .or.
     &      (iroute .eq. 1 .and. iand(imdctl(1),64).eq.0)) then
          ontim = trptim(n)
        else
          ontim = trptimss(n)
        endif
        rtrip = ontim .ge. 0.0
        if (.not.rtrip .and. ltrip) then
          ontim = timehy
        else if (iand(ntrpop(n),2).eq.0 .and. rtrip .and. .not.ltrip)
     &  then
          ontim = -1.0                           Section 2.5
        endif
        if (iroute .ne. 1 .or.
     &      (iroute .eq. 1 .and. iand(imdctl(1),64).eq.0)) then
          trptim(n) = ontim
        else
          trptimss(n) = ontim
        endif
        n = n + ntlskp
      enddo
c
  100 timehy = timx
      return
      end
```

**Figure 5a. A portion of a pre-transformed subroutine**

```
      subroutine logic_trip
!  Tests logical trip conditions; sets trip conditions and time-of-trip.
      use ctrlmod
      use trpmod
      implicit none
      integer   :: i
      real(sdk) :: ontim, timehy, timx
      real(sdk), pointer :: triptime
      logical   :: ltrip, rtrip
!
!  Executable code
!  1.0  Initialization
      timx = timehy
      timehy = timehy - dt
!  2.0  Logical trips.
      do i = 1, ntrpl
!  2.1  Evaluate left and right sides logical trip
        ltrip = trplp(i)%time >= 0.0 .neqv. btest(trpl(i)%ntrpop, 4)
        rtrip = trplp(i)%time2 >= 0.0 .neqv. btest(trpl(i)%ntrpop, 5)
!  2.2  Logically combine sides
        select case (ibits(trpl(i)%ntrpop,24,8))
        case (1)
          ltrip = ltrip .or. rtrip
        case (2)
          ltrip = ltrip .and. rtrip
        case (3)
          ltrip = ltrip .neqv. rtrip
        end select
!  2.3  Time trip was true
!  In latching trip case, this is the time it first became true.
!  Otherwise, this is the previous advancement time.
        if (iroute /= 1 .or. btest(imdctl(1),6)) then
          triptime => trpl(i)%trptim
        else
          triptime => trpl(i)%trptimss
        endif
        ontim = triptime
        rtrip = (ontim >= 0.0)
!  2.4  Recalculate time of trip status
        if (.not.rtrip .and. ltrip) then
!  2.4.1  Time turns on
!  Trip was false on previous advancement but is now true
          ontim = timehy
        else if (.not.btest(trpl(i)%ntrpop, 1) .and. rtrip .and.
     &   .not.ltrip) then
!  2.4.2  Trip goes false
!  Non-latching, trip value was true & trip condition is now false
          ontim = -1.0
        endif
!  2.5  Record trip status
        triptime = ontim
      end do   !i
!
!  3.0  Reset transient cumulative time.
      timehy = timx
      return
     end subroutine logic_trip
```

**Figure 5b. Same subroutine portion transformed**

## CONVERSION TESTING

During the reformulation task, other development projects were proceeding simultaneously. These sometimes seriously impacted the conversion project. Moreover, conversion errors impacted code users. Thus testing was very important to ensure that the code continued to produce correct calculations.


Copyright © 2010 by ASME

To ensure that RELAP5-3D worked properly for users during the transformation process, stringent testing was performed frequently. A second reason frequent testing was efficiency in error finding. It is quickest and easiest to locate an error among a small set of recently converted subprograms.

Testing employed a suite of test problems and an evaluation *metric* for detecting errors.

### Testing Procedure

- Generate output with test case i with updated code.
- Generate output with test case i with untransformed code.
- Compare output with the Linux diff utility.
  - Eliminate differences due date, time-stamp, CPU time and memory mapping.
- If more than Z differences remain, reject the update.

Apply the test procedure to all standard test cases. If all cases are acceptable, the update is acceptable.

Zero was the value of Z for purposes of source code conversion. Thus, *the output of each test case must be character for character exactly the same, before and after conversion, on the "printed output" file*. The testing procedure was applied to both update sets and completed versions.

A limitation of this test procedure is that it does not check all decimal places of the calculations, about 14 for floating point values, but rather only those printed. An improvement would be to increase the accuracy of the printed output or to test binary output. A second limitation of this test procedure is the impact of other development upon calculations being compared. If recent developments or error corrections affect calculations of one or more test cases, the responsible developer must determine the correct values. This complicates the test procedure.

Finally, this test procedure cannot find all possible errors unless the test suite has 100% coverage of all lines of source code being transformed. Coverage is detailed in Section 6. The extent of the coverage of the test suite determines the effectiveness of the test procedure. For the FORTRAN 95 conversion, some new cases were added to test additional portions of the code thereby increasing coverage.

## RESULTS AND MEASUREMENTS

The reformulation of RELAP5-3D has resulted in numerous improvements. Section 6 presents a variety of quantitative measures and interpretations of those measurements. Results of reformulation include both longevity improvements, such as legibility and maintainability, and code feature improvements, such as machine independent binary output. Two principle categories of evaluating the reformulated program and comparing it with its predecessor are static and dynamic measurements. Static measurements include source code analyses such as size, percentage, complexity measures, and number of user problems fixed. Dynamic measurements include code run speed, coverage, and other statistics relating to the performance of the program as it runs.

## REFORMULATION LONGEVITY RESULTS

Longevity changes are those upgrades and reformulations made to prevent obsolescence. Many changes were made to increase longevity of RELAP5-3D. These include modernization of the database and source code, use of a modern programming language and its constructs, code complexity reduction, legibility increase. These lead to development and maintenance cost reduction.

<u>Code modernization</u> was carried out in the reformulation of the database and source code reported in Sections 2 through 4. The modern language is FORTRAN 95. All reference to language features listed in the obsolescent or downgraded language features of the handbook have been eliminated. Older constructs, such as arithmetic if, assigned go to, indexed go to, alternative returns, buffer statements, and the like have been eliminated or replaced with modern constructs such as if-then-else and case statements. Older programming paradigms have been replaced by modern ones. For example, equivalence was replaced by derived types, mapping multi-dimensional data into linear arrays was replaced by natural declaration and indexing of multi-dimensional quantities, and pointers via indexing in a container array replaced by true pointers. These have greatly simplified the source code and eliminated sources of error.

<u>Portability</u> was addressed in the preliminaries box of Figure I. Many machine-specific library calls, such as bit-packing, time and absolute memory address functions, were replaced or eliminated by FORTAN 95 intrinsic functions or pointers. A new feature, machine-independent binary files, has increased portability so that files produced on one platform can be used on another, regardless of hardware or operating system. An important portability issue was adaptation to the Linux Operating System.

<u>Code readability</u> or legibility was increased in many ways. First, the code was restructured [6]. Structured programming is both highly modular and easier to read and understand. Every block of code has only one entry point and one exit point. There are no backward jumps except as part of a loop construct. Second, dead code was eliminated. Third, source code formatting rules were applied uniformly. Fourth, more internal documentation has been inserted and outdated comments replaced. Fifth, some subprograms were re-factored; that is, repeated code and sections of pre-compiler protected code were moved into internal "contained" subprograms. Finally, some subprograms were fully rewritten as needed to simplify, clarify, or implement the algorithm with modern programming constructs.

The result of modernization and readability changes, and the restriction to just one brand of compiler, has reduced <u>maintenance and development</u> costs. Simplification of indexing has reduced the potential for indexing errors. Legibility increase reduces the time required to perform development and to find coding errors. Moreover, FORTRAN 95 has a form and structure that is easy for a C++ or Java programmer to learn, as already demonstrated with INL summer intern Wang [16] who started with no FORTRAN experience. Thus the pool of potential programmers has been expanded markedly over


Copyright © 2010 by ASME

FORTRAN 77, and this, along with the aforementioned changes, will increase the longevity of RELAP5-3D.

## IMPROVED FEATURES AND ASSESSMENT

Some of the new features implemented in reformulating RELAP5-3D have been more operating system accessibility, split restart and plot files, machine independent output files, and expandable database. A developmental assessment (DA) [17] was performed on the reformulated version using 100 representative test cases. Table I compares the features and assessment of the reformulated of the previous versions.

As a result of the reformulation work, RELAP5-3D now installs on Windows XP platforms with 64-bit Fortran compilers and passes its suite of test cases on Linux, Unix (Solaris), and Windows XP. Only one compiler is currently supported, but a number of new compiler levels are supported.

Another new feature is the split restart-plot file. Instead of a single restart-plot file, there are two separate files, a restart file and a plot file. There is now no need to post-process the restart-plot file to access the plot data. That data is already separated into its own file for immediate use by the code user.

The introduction of eXtended Data Representation (XDR) binary output files has created new possibilities for RELAP5-3D code users. It does not matter if the hardware is big or little endian, the XDR binary data is the same. Thus the files can be written on one platform, say Linux, and used on another, completely different, platform, such as Windows. The fluid properties, plot, and strip are all written in machine independent form. The restart file is not written in machine independent binary. The fluid and plot files are written in machine independent XDR format, but the strip file is in ASCII only.

**Table I. Comparison of Old and Reformulated Code Features**

| Category | Pre-F95, Version 2.4.1.2 | F95 Reformulated Version 2.9.3 |
|---|---|---|
| **Compilers** | Many, older | Intel Fortran 9.1 and 10.1 |
| **Modularity** | Unstructured | Strongly Modular – Structured |
| **Dead Code** | Many unused source files | Removed 162 unused files |
| **Platforms** | Windows, Unix | Linux, Windows, Unix |
| **Portability** | O/S specific bit, time, loc. utilities | Fully portable F95 intrinsic library |
| | Binary machine *dependent* files | XDR binary machine *in*dependent |
| **File Form** | Combined restart-plot file | Separate restart and plot files |
| **Plot format** | N/A | ASCII or XDR binary |
| **Memory** | Upper-limited | Expands to fit model |
| **DA** | No | Yes |

Another new feature is ASCII plot files. The user may specify in the code input file that the plot file is to be written in the ASCII format. This allows the plot file to be immediately imported into application plot programs for immediate use; there is no need to post-process.

The DA is a form of validation that is applied to a subset of test cases of interest. The DA tests were comprised of separate effects, integral effects, and plant models for which data was available for comparison. The calculations are considered acceptable within the engineering judgment standards described in the DA [17].

During the transformation project, code development, in the form of additional physical models and resolutions to user problem reports, was ongoing. Because these enhancements affect the calculations, comparison between pre- and post-F95 versions can be made for few input models; however, those show excellent agreement between the old and new versions.

## STATIC MEASUREMENTS

Static measures of the code include counts and percentages of important programming aspects. These aspects include comments and complexity measures. These measures can be applied across the whole code as a count or an average or as a best or worst case. Improvement due to the reformulation can be measured by comparing reformulated version 2.9.3 against the standard Pre-FORTRAN 95 version, 2.4.1.2. The static measures are compared in Table II.

The readability measures are confined to ratios of comment to non-comment lines. This ratio would increase if executable statements rather than non-comments were considered. This statistic does not measure the quality of comments, but does clearly indicate that the number of files with significant percentage of comments increased by a factor of over 2.5. This was due to imposing certain standards for comments on the newly written code, such as modules.

**Table II. Static Source Code Analysis Measures**

| Category | 2.4.1.2 | 2.9.3 |
|---|---|---|
| **Readability** | | |
| Best comments to code ratio | 6.64 | 32.2 |
| Files w/ comments to code ratio >= 0.3 | 390 | 954 |
| **Complexity** | | |
| Maximum cyclomatic number | 982 | 460 |
| Files with cyclomatic number >= 100 | 68 | 13 |
| Maximum Nesting (levels deep) | 20 | 15 |

The McCabe Cyclomatic index is the industry standard for measuring code complexity. It measures the number of independent linear paths through a section of source code, and also indicates the number of branches within that same piece of source code. The higher the index number, the more branches are in the code, which in turn affects the quality of the testing and the maintainability of the code. Though opinions vary, subprograms with an index greater than 100 are considered complex, and anything over 500 is generally considered too complex.


Copyright © 2010 by ASME

The number of complex subprograms dropped from 68 to 13, a factor of 5. There are no subprograms over 500 in the transformed code.

Nesting counts the sub-blocks of blocks. A loop has one interior block; a branching construct, such as case or if-then-else statement, has several at the same level. These blocks may contain loop or branching construct, their blocks are sub-blocks at level two. Nesting counts the number of levels to which the sub-blocking goes. Considerable improvement was obtained in nesting.

**Table III. User Problem Resolution**

| | Reports | Resolved | After 2.4.1.2 | Fixed by F95 |
|---|---|---|---|---|
| **1998** | 85 | 31 | 1 | 1 |
| **1999** | 75 | 50 | 1 | 1 |
| **2000** | 92 | 47 | 2 | 2 |
| **2001** | 86 | 67 | 3 | 3 |
| **2002** | 86 | 72 | 0 | 0 |
| **2003** | 61 | 44 | 2 | 2 |
| **2004** | 62 | 39 | 7 | 7 |
| **2005** | 55 | 32 | 30 | 2 |
| **2006** | 71 | 36 | 35 | 1 |
| **2007** | 51 | 27 | 23 | 4 |
| **2008** | 47 | 23 | 18 | 5 |
| **2009** | 42 | 10 | 10 | 0 |
| | 813 | 478 | 132 | 28 |

It must be recognized that many user problems extant in the older version were resolved in the newer. Table III shows User Problems (UP) reported from the first version of RELAP5-3D through version 2.9.3.

Of the 478 UP that have been resolved, 132 solutions came after version 2.4.1.2 and are in version 2.9.3 only. These include some 28 that were resolved merely by the reformulation into FORTRAN 95.

## DYNAMIC MEASUREMENTS

Dynamic measurements are made by executing the program and collection statistics about the run. There are two primary dynamic measurements, namely, coverage analysis and run speed. Coverage spied upon the code as it runs and collects information about what part of the code is executing. It is important to have the test suite exercise as much of the source code as possible so that coding errors can be found by the test suite and not the users. On the other hand, code users need the code to run a quickly as possible so that they can perform their analyses in a more timely fashion; therefore, measures of code run speed are very important also. Coverage analysis is presented in Table IV and run speed in Table V.

As can be seen, the number of test cases for version 2.9.3 has grown immensely, though the DA and PVM test cases increased through other development projects.

The coverage analysis can be broken down by subroutines or by groups of subprograms. Table IV shows subroutine groupings according to their location by folder/directory. The relap directory holds most of the essential subprograms that implement the physics and other calculations described in the RELAP5-3D manuals. The envrl directory holds those that provide service, such as linear equation solution, table look-ups, interpolations, and input control service.

**Table IV. Comparison of Test Cases and Coverage Percentages**

| Category | Pre-F95 Version 2.4.1.2 | | F95 Version 2.9.3 | |
|---|---|---|---|---|
| **Test Cases** | 188 | | 2034 | |
| **Product Release** | 188 | | 221 | |
| **PVM Dt Tests** | 0 | | 1760 | |
| **DA** | 0 | | 53 | |
| **Coverage Analysis** | Files | Stmts | Files | Stmts |
| **Relap Directory** | 63.87 | 44.72 | 80.37 | 61.51 |
| **Envrl Directory** | 35.46 | 38.91 | 54.24 | 51.91 |

The percentages are files, functions and statements. For files, the percentage is the number of files entered during execution of any of the test suite cases divided by the number of files. Similarly for function and line, the count is for the number of subprograms in any file entered divided by the total number of such functions; lines are limited to executable lines.

A large improvement of 14, 15, and 17 percent can be seen in the coverage of the files, functions and lines respectively of the relap directory. In the envrl the improvements are 19, 19, and 23 percent In both cases, the coverage is now over 50 percent. It should be noted that only the standard test suite is used for these measurements. Neither the Developmental Assessment nor the new 1760 problem suite provided by the other PVM project are employed in collecting these statistics.

**Table V. Run speed indicators for AP600 test cases**

| | Pre-F95 Version 2.4.1.2 | | |
|---|---|---|---|
| **Model** | **Attempts** | **CPU (sec)** | **CPU / Attempt** |
| **PMPS** | 139 | 12.26 | .0882 |
| **PWRS** | 455 | 34.28 | .0753 |
| **SBS** | 419 | 31.74 | 0.758 |
| | **F95 Reformulated, 2.9.2** | | |
| **PMPS** | 139 | 12.46 | .0896 |
| **PWRS** | 519 | 33.36 | .0643 |
| **SBS** | 419 | 28.08 | .0670 |

CPU time on a SUN platform with an Opteron chip, using SUSE Linux 9.1 and an Intel 9.1 compiler was used to collect


Copyright © 2010 by ASME

these samples. The values will vary from platform to platform. For some smaller and shorter running problems, the reformulated code's run timings may are smaller while for others they are larger; no reason has yet been determined for this. However, for larger problems, the reformulated version 2.9.3 generally runs faster or at least comparable to the standard version 2.4.1.2.

The problems all use semi-implicit time advancement, the first is a pump transient, the second a simple run to steady state, and the last a small break. The number of attempts is the same in the first and third and larger for the new code in the second. Yet the CPU time goes up very slightly, less than 2%, for the first problem, while it actually goes down for the third and, despite the increased number of advancements, for the second as well. The grind time, CPU seconds per attempt, is significantly better for the second and third run while it is comparable for the first.

## SUMMARY

The Reformulation of RELAP5-3D is complete. It includes a complete transformation of its database and conversion of its source code from FORTRAN 77 to FORTRAN 95. The code runs all the test suite problems that it ran before, plus over 1800 more test cases. Its longevity has been increased by its reformulation in modern language, reduction in complexity, and restructuring into a highly modular form that is more readable, and less time consuming to develop and maintain. Virtually every measurement of code improvement shows the new version has more capabilities, is more portable, runs as fast or faster, and is better tested.

## ACKNOWLEDGMENTS

This 4.5 year project was funded by the US DOE through a variety of projects. Thanks are given to the contributors to the source code conversion; this includes Dr. Richard Riemke, Richard Wagner, Dr. Walter Weaver, and Nolan Anderson. Appreciation is given to participants in the source code conversion and preliminaries; this includes Hope Forsman, Richard Moore, Peter Cebull, Cliff Davis, Dr. Paul Murray, Joshua Hykes, Dr. Donna Guillen, and Riley Cumberland.

## COPYRIGHT STATEMENT

This manuscript has been authored by Battelle Energy Alliance, LLC under Contract No. DE-AC07-05ID14517 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a nonexclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes.

## REFERENCES

1. RELAP5-3D Development Team, *RELAP5-3D Code Manual, Volume 1*, INEEL-EXT-00834, Revision 2.4, Idaho National Laboratory, Idaho Falls, ID (2005).
2. J. C. Adams, W. S. Brainerd, J. T. Martin, et al., Fortran 95 Handbook, MIT Press, Cambridge, MA, USA (1997).
3. G. L. Mesina, *LPCIS N-16 Lobe Power Calculation, Theory, and Programming*, Engineering Design File TRA-ATR-1821, Project File Number 021008, Idaho National Laboratory, Idaho Falls, ID (2002).
4. J. Mahaffy, J Uhle, and J. Dearing et al., Architecture of the USNRC Consolidated Code, Proc. ICONE-8 April, Baltimore, USA (2000).
5. G. L. Mesina, "Architectural Advancements in RELAP5-3D," Proceedings of the 2005 ANS Winter Meeting, November 12-17, 2005.
6. D. P. Guillen, G. L. Mesina, J. M. Hykes, "Restructuring RELAP5-3D for Next Generation Nuclear Plant Analysis," *2006 Transactions of the American Nuclear Society*, Vol. 94, June 2006.
7. G. L. Mesina, J. M. Hykes and D. P. Guillen,, "Streamlining RELAP5-3D", Proceedings of NURETH-12, Pittsburgh, PA, Nov, 2007.
8. J. H. McFadden, et al., "RETRAN-03 Code Manual," *EPRI NP-7450*, May, (1992).
9. C. W. Stewart, et al, "COBRA-IV: The Model and the Method," *BNWL-2214*, Pacific Northwest Laboratory, (1997).
10. J.D. Ramshaw and C.H. Chang, Computational Fluid Dynamics Modeling of Multi-component Thermal Plasmas, *Plasma Chem. Plasma Process.* **12** (1992), pp. 299–325.
11. P. J. Turinsky, et al., "NESTLE: A Few-Group Neutron Diffusion Equation Solver Utilizing the Nodal Expansion Method for Eigenvalue, Adjoint, Fixed-Source Steady-State and Transient Problems," *EGG-NRE-11406*, Idaho National Engineering Laboratory, Jun, (1994).
12. K. 0. Pasamehmetoglu, J. Spore, et al., "TRAC-PFl/MOD2, Theory Manual," Los Alamos National Laboratory Report, LA-12031-M, Los Alamos National Laboratory, Los Alamos, NM, USA, (1993).
13. T. J. McCabe. "A Complexity Measure," *IEEE Transactions on Software Engineering*, **Vol. SE-2, No. 4**, pp. 308-320, (1976).
14. O. J. Dahl, E. W. Dijkstra, C. A. Hoare, *Structured Programming*, Academic Press, London, England (1972).
15. D. E. Knuth, "Backus Normal Form vs. Backus Naur Form," *Communications of the ACM* **7 (12)**, pp. 735–736 (1964).
16. Raymond Wang and G. Mesina, "Implementation of Viscous Effects in RELAP5-3D Thermal Hydraulics Code," *The Journal of Undegraduate Research of the DOE Office of Science*, submitted Aug 4, 2009.
17. Paul D. Bayless, et al, "Developmental Assessment of RELAP5-3D Version 2.9.2," *INL/EXT-09-15965*, Idaho National Laboratory, (2009).

 Copyright © 2010 by ASME