Technical Report CS/E 85-010 May 1985

PARALLEL PROCESSING ON THE DENELCOR HEP WITH LARGE GRAIN DATA FLOW TECHNIQUES

Robert G. Babb II
Lise Storc
Oregon Graduate Center
19600 N.W. Von Neumann Drive
Beaverton, OR 97006

(503) 690-1151

PARALLEL PROCESSING ON THE DENELCOR HEP

WITH LARGE GRAIN DATA FLOW TECHNIQUES[†]

Robert G. Babb II
Lise Storc
Department of Computer Science and Engineering
Oregon Graduate Center
19600 NW Von Neumann Dr.
Beaverton, OR 97006

503-690-1151

Final Report

for

Los Alamos National Laboratory
Computer Research and Applications Division (C-3)

and

DENELCOR, Inc.
Aurora, Colorado

30 April 1985

[†] This work was supported by Denelcor under Contract 84-S-010, and Los Alamos National Laboratory under Contract 9-Z-34-P-3915-1.

Table of Contents

1. INTRODUCTION

This report summarizes the results of research and software tool development aimed at making parallel processing much easier and less error-prone. The approach we have demonstrated involves developing scientific application programs using a high-level parallel model (Large-Grain Data Flow or LGDF). Programs written in the LGDF framework have been transformed into appropriate source code for a variety of sequential, vector, and parallel machines[1] [2] [3].

We based the tool development for this project on a set of prototype LGDF implementation tools based on macro-expansion techniques. As part of the current project, these existing software tools were modified to generate parallel scheduling mechanisms that implement the LGDF model efficiently on the HEP. To demonstrate the use of the tools, for parallel processing, we transformed a sequential Monte Carlo code (GAMTEB) into an LGDF code capable of either sequential or parallel operation via a series of transformation steps.

Section 2 of this report presents an overview of the Large-Grain Data Flow model, and the way it was implemented on the HEP.

Section 3 discusses our adventures in getting the original sequential GAMTEB to run correctly on a VAX running Berkeley 4.2 Unix. (This proved to be a surprisingly difficult job, and in fact was more difficult than parallelizing the sequential code

after it was runnable on the VAX).

In Section 4, modifications to sequential GAMTEB in anticipation of parallel operation are discussed. These modifications allowed more highly parallel operation, and resulted in a single process code directly comparable to the LGDF multi-process version.

In Section 5, we present the rationale for the LGDF partitioning of the processes and data structures within GAMTEB, as well as a discussion of some minor modifications to the code required to meet LGDF standards. We also summarize our experiences in debugging the LGDF (sequential) code.

In Section 6, we discuss some further changes and bugfixes in the LGDF macros that became apparent when we actually ran the LGDF model in parallel under the HEP Unix Parallel Executive (UPX). The problems were not "GAMTEB" problems, but LGDF macro problems caused by the shift to HEP/UPX. They were not difficult to fix, but tracking them down in a parallel program was, as usual, quite difficult.

Section 7 presents a summary of our results and recommendations for further research.

2. OVERVIEW OF LARGE-GRAIN DATA FLOW

The Large Grain Data Flow (LGDF) approach is based on viewing programs as being made up of systems of data-activated processing

units. Using a coherent hierarchy of data flow diagrams, complex systems are specified as compositions of simpler systems. The lowest level processing units correspond roughly to FORTRAN SUBROUTINE's[*].

The steps involved in modeling and implementing a FORTRAN code using the LGDF computation model and tools are as follows:

1) Draw Data Flow Diagrams- create a hierarchical, consistent set of system data flow diagrams that express the logical data dependencies of the program or problem modeled.

2) Create Wirelist- encode the data flow dependencies of the set of data flow diagrams using macro calls.

3) Package Data Declarations- identify and cluster the FORTRAN data declarations corresponding to each data link in the system data diagrams. (These will become a set of labeled COMMON declarations in the generated programs).

4) Add Data Flow Control to Program Fragments- embed standard data flow control macro calls in the FORTRAN code.

---

[*] These lowest level programs are "large-grained." This means that the amount of processing that a lowest level routine performs each time it executes is large compared to the overhead of scheduling its execution. On the DENELCOR HEP computer the mechanism needed for data-activated execution scheduling is built-in to the hardware, allowing efficient execution even for "fine-grained" programs.

5) Expand Data Flow Macros- macro expand the wirelist, packaged data declarations, and program fragments to produce compilable FORTRAN for the particular machine desired (in this case, either the VAX Unix 4.2, or the HEP/UPX) [*].

6) Compile and Execute- including if desired, pre- and/or post-compilation optimization steps available for the particular target environment.

More details on Large-Grain Data Flow and its implementation on the HEP can be found in [4] [5] and in the paper[6] (a preprint of a book chapter written as part of this work, included in this report as Appendix A).

3. PORTING SEQUENTIAL GAMTEB TO THE VAX

The changes made to the original, sequential version of GAMTEB to get it to compile on the VAX (running under Berkeley 4.2bsd UNIX with the f77 compiler) were of several major types:

1) changes for FORTRAN77
2) VAX-specific changes
3) Library subroutine differences

---

[*]We use the standard Unix general purpose macro processor "m4" for the current version of the LGDF macros. For HEP-OS, programs were macro expanded under Unix, and the resulting source code transmitted to the HEP for compilation and execution. The macros have now also been run successfully under HEP/UPX.

The next step was to get GAMTEB running correctly. This involved tracking down and fixing a serious bug caused by the equivalencing of reals and integers. Finally, a portable random number generator was installed to insure that results would be identical on the HEP and the VAX.

A summary of the changes is given below. The notation [original-line#] refers to line numbers in the original, sequential GAMTEB (Appendix B). The reference [f77-line#] is to line numbers in the sequential version of GAMTEB as modified to get running under 4.2 (Appendix C). The notations [lgdf-d# line#] [lgdf-p# line#] refer to the Large-Grain Data Flow version (Appendix D):

1) The PROGRAM line was changed from:

```
      PROGRAM GAMTEB(OUTPUT,TAPE4=OUTPUT)          [original-3]
```

to:

```
      PROGRAM GAMTEB                               [f77-4]
```

2) Non-standard multiple assignment statements, for example:

```
      IF(J.EQ.JA) D2=D1=-2.*A1                     [original-373]
```

was changed to:

```
      IF(J.EQ.JA) THEN                             [f77-427 to 430]
         D1=-2.*A1
         D2=-2.*A1
      END IF
```

(see also [original-403] and [f77-461 to 464]).

3) The machine-dependent subroutine 'SECOND' was replaced by a dummy version:

```
SUBROUTINE SECOND (T)                                    [f77-573 to 576]
T=0
RETURN
END
```

4) The following variable names were reduced to six characters:

```
ESCAPE2 --> ESCAP2          see for example  [f77-24]
ESCAPEI --> ESCAPI                           [f77-25]
RESCAPE --> RESCAP                           [f77-27]
```

5) The number of particles was reduced so that GAMTEB would run on the VAX in a reasonable length of time. That is,

```
NPP=500000                                               [original-46]
```

was changed to:

```
NPP=100                                                  [f77-67]
```

6) "Almost underflow" exponents -123 to -37 were changed to fit into the exponent range of the VAX F_floating data type (32 bit), for example:

```
IF(XPP(I).EQ.0.) XPP(I)=1.0E-123                         [original-38]
```

was changed to:

```
IF(XPP(I).EQ.0.) XPP(I)=1.0E-37                          [f77-57]
```

(see also [original-39] and [f77-59]).

7) A real variable was initialized with an integer constant:

```
      FIM(1)=1                                          [original-72]
```

and for stylistic reasons was changed:

```
      FIM(1)=1.0                                        [f77-103]
```

8) An extra initialization of the variable INBNK was removed:

```
      INBNK=0                                           [original-74]
      changed to:
C     INBNK=0                                           [f77-106]
```

9) The real-integer equivalencing problem related to banking particle values was fixed. In the original version of GAMTEB, a real array (BANK) and an integer array (IBANK) were equivalenced. Another array (PBL) was equivalenced to a series of variables in COMMON which included two integers (IA and NP):

```
COMMON X,Y,Z,U,V,W,ERG,IA,WT,NP                    from [original-8]
DIMENSION BANK(100,10),PBL(10),IBANK(100,10)       from [original-12]
EQUIVALENCE(PBL,X),(BANK,IBANK)                          [original-14]
```

This bug showed up because when moving a floating point number, the VAX hardware changes an F_floating datum with an exponent value of 0 (bits 14:7) and a sign bit of 0 (bit 15) to the value 0 by zapping any other "dirty zero" bits. Thus,

real to real assignments between arrays equivalenced to integer values often resulted in the loss of the integer values!

The problem was solved by creating new arrays and only equivalencing arrays of the same data type, and then modifying the pertinent sections of the code:

```
COMMON X,Y,Z,U,V,W,ERG,WT,IA,NP                          [f77-9]
DIMENSION BANK(100,8),PBL(8),IBANK(100,2),IPBL(2)        [f77-29]
EQUIVALENCE(PBL,X),(IPBL,IA)                             [f77-30]
```

(See also [f77-197 to 200, 204 to 207, and 209 to 210]).

NOTE: This was a very difficult problem to track down.

10) Random number generators. GAMTEB was first run with a dummy random number generator which always returned the constant 0.5. (By the choice of the constant 0.5 it was also discovered by accident that any random number generator that returns two consecutive 0.5's will cause GAMTEB to blow up in subroutine ISOS during the computation of T3 (!):

```
T1=2.*RANF(KRN)-1.                                [f77-528 to 533]
T2=2.*RANF(KRN)-1.
RSQ=T1**2+T2**2
U=2.*RSQ-1.
T3=SQRT((1.-U**2)/RSQ)
```

11) Next, GAMTEB was run using the UNIX random number generator RAND. Later, a more portable random number generator (suggested by Paul Frederickson) was installed:

```
REAL FUNCTION RANF(KERN)                                    [f77-584 to 588]
KERN = MOD(1+9621*KERN,131072)
RANF = FLOAT(KERN)/131072.
RETURN
END
```

12) The resulting program was run on both the VAX and the HEP (as a sequential code for 100 particles, producing the same output.

4. MODIFICATIONS TO SEQUENTIAL GAMTEB FOR PARALLEL OPERATION

This section describes changes to the original GAMTEB necessary to allow deterministic parallel computation of the histories of separate particles and their descendants. This was done with the addition of a second random number sequence generator. In the modified method, new particles are started with random seeds from the first sequence, and the history of that particle and its daughter particles is simulated using that seed and the second random number generator.

The rationale for this is as follows. Using only one random number generator results in an inherent sequential bottleneck. If we try to run particle histories in parallel, each process concurrently running a history would be competing for the next random number. In order to have a deterministic program, a process would have to completely finish its particle history before the next process could be allowed to compete for its random

numbers. In any case, the potential parallelism would be greatly reduced.

1) In the original GAMTEB, the state of the random number generator was kept inside the random number generation routine RANF. The value of the argument to RANF, KRN, was not used as an input to the random number generator. (It relied on its retained value from its last use) as in:

```
S = -ALOG(RANF(KRN))/XST                                [original-121]
```

This method causes problems when calling the random number generator from a variety of subroutines and functions that do not "remember" the last random number generated. Therefore, the returned value was also placed in COMMON to communicate this "latest value":

```
COMMON /NEW/ KRN                                        [f77-17]
```

This only needed to be done for "KRN", the returned value of the added (second) random number generator, because all calls to the first random number generator occur in the same routine.

(see also [f77-404,483,526 and 552])

2) A second portable random number generator was added:

```
REAL FUNCTION RANDO(KERN)                          [f77-578 to 582]
KERN = MOD(1+7421*KERN,131072)
RANDO = FLOAT(KERN)/131072.
RETURN
END
```

This function provides a seed for the random numbers generated by RANF (see above) that are needed to run one particle history.

(see also [original-55], [f77-81], and [f77-133 to 135])

The resulting program was run on both the VAX and the HEP with 100 particles and yielded equivalent results (Appendix F).

5. DEVELOPMENT OF A PSEUDO-PARALLEL LGDF VERSION OF GAMTEB

The first step in creating the LGDF model of GAMTEB was to identify problem constants. The constants were initialized in the original version both via a BLOCK DATA subprogram (for variables in labeled COMMON) and via assignment statements. These were identified as the following datapaths in the LGDF model:

```
d03-(problem constants)                            [f77-13, 19, 33 to 37]
d04-(converted cross section tables)               [f77-21, 39 to 51]
d05-(source values)                                [f77-9]
```

Statements that initialize these values were packaged as the LGDF process:

p10-(set up problem constants) [f77-53 to 63, 70 to 71, 77, 83 to 86, 102 to 104, 121 to 129, and 131]

The LGDF network with the result of this level of data flow analysis for GAMTEB is shown in Fig. 1.

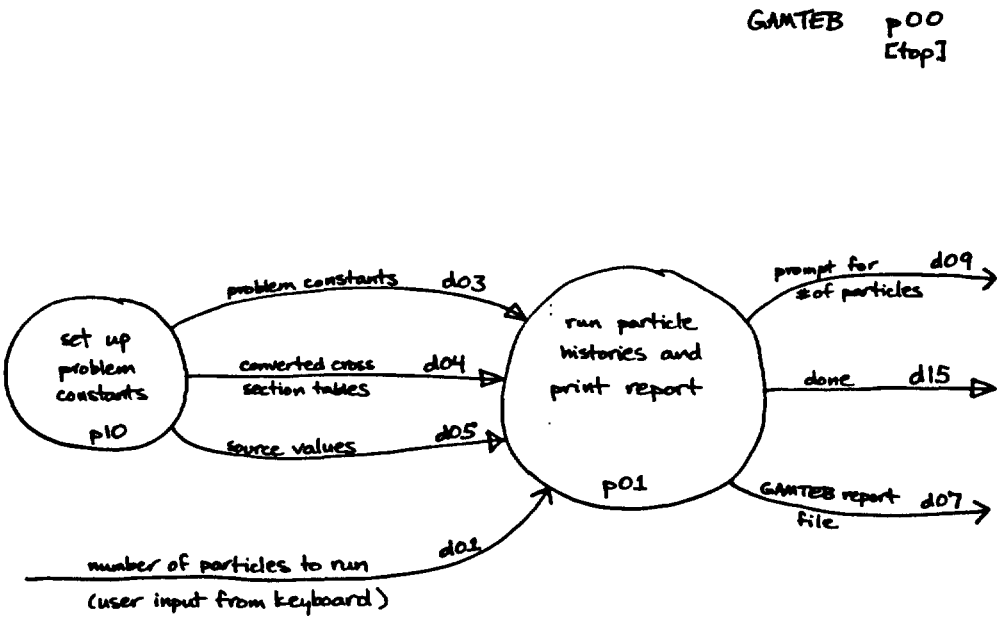


Fig. 1. An LGDF network for p00-(GAMTEB).

In the original GAMTEB, the number of particle histories to be run was initialized via an assignment statement:

```
C         INITIALIZE PROBLEM INPUT                    [original-45 to 46]
      NPP=500000
```

We decided as part of the improvements we were making to allow this value to be specified interactively. The prompt for user input and the user response are shown on Figure 2 as d09-(prompt for # of particles) and d01-(# of particles to run). (The code for the LGDF process p12-(init GAMTEB run parameters) as well as the other process and datapath definitions can be found in Appendix D.)

To assist in running particle histories independently, we divided the particle statistics accumulators into two groups, global and local. The global accumulators are shown on Fig. 2 as d06-(run statistics), and these are initialized once per run by p12.

The process p13-(set up for particle counting) initializes various counters for use by other LGDF processes. The counters are part of the LGDF datapaths:

d11-(particle completion control)
d12-(particle start control) and
d10-(particle count for report)

The LGDF process p13 also initializes the value (KRN2 on d12) used to seed the first random number generator (RANDO in p16.m).

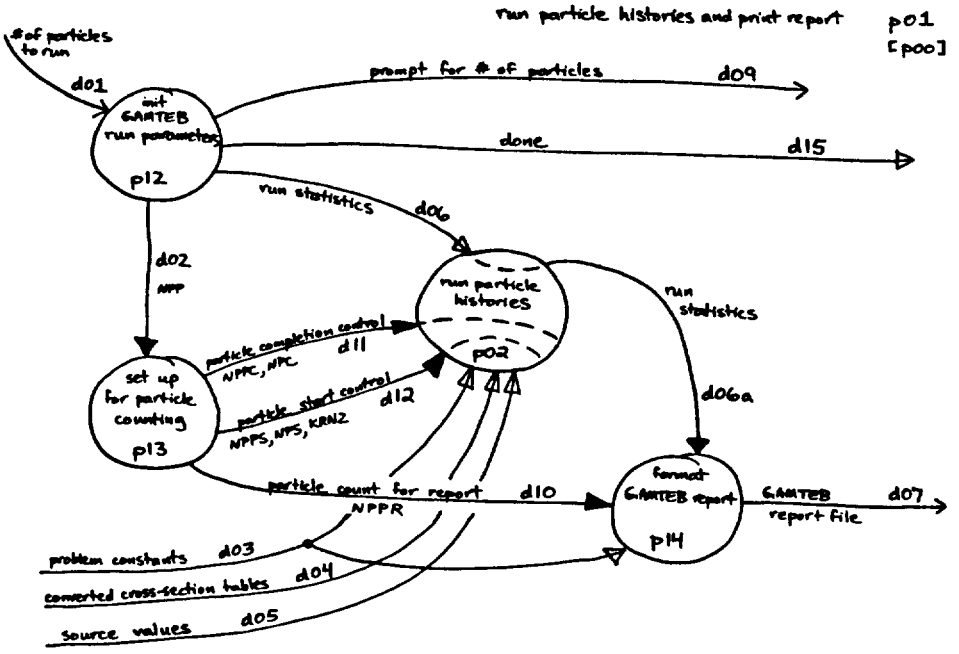


Fig. 2. LGDF network for p01-(run particle histories and print report).

The statements that generate the report produced by GAMTEB (see [f77-294 to 295, and 298 to 389]) have been packaged as p14-(format GAMTEB report). The process p14 does some summary computations, and formats and writes the report based on the final run statistics (d06a).

The actual work of running particle histories involves three LGDF processes:

p16- (generate random seed for next particle)
p17- (run history for one particle and descendants) and
p18- (check for completion)

The LGDF network for this level is shown in Fig. 3.

The process p16 uses the function RANDO to generate a random seed for use by the second random number generator (RANF) in running the next particle history.

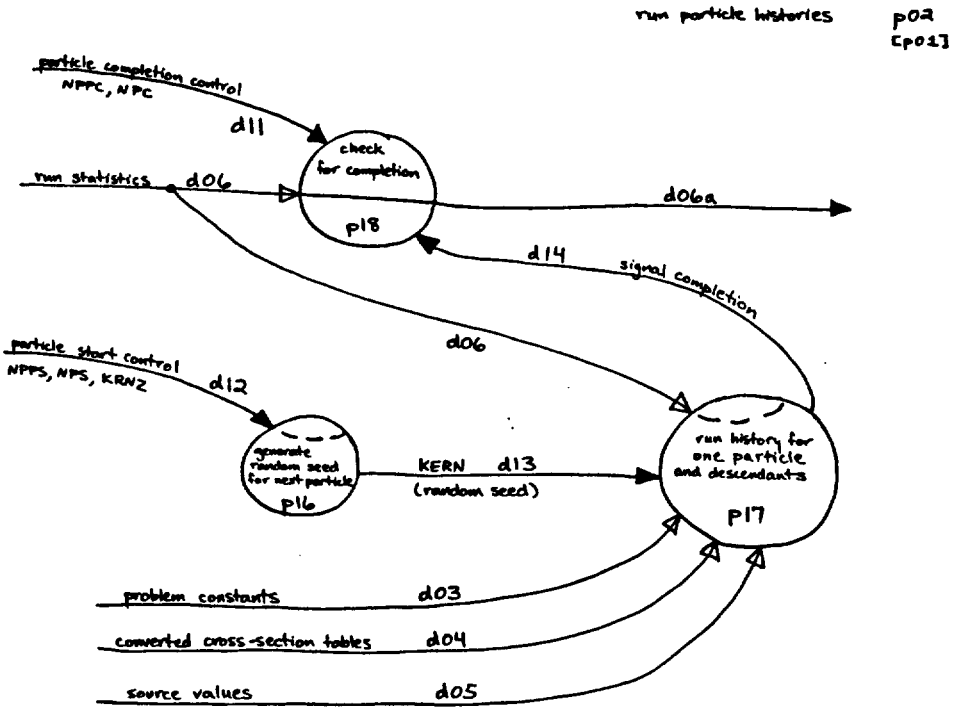


Fig. 3. LGDF network for p02- (run particle histories).

The bulk of the original GAMTEB code (approx. 300 lines) was put mostly unchanged into p17-(run history for one particle and descendants). This process simulates the history for a particle and its daughter particles, using a local particle bank (BANK(100,8) and IBANK(100,2)). It also makes local copies of the initial source values in d05 to avoid extensive variable renaming within p17. The random seed (d13-(KERN)) received from p16 is also copied into a local variable to allow p16 to generate the random seed for the next particle as soon as possible. This does not aid sequential processing, but is the key idea that allows overlapped (parallel) computation between independent particle histories. Run statistics for each new particle and its descendants are kept in variables local to p17.

The only other major addition to the original GAMTEB code required was code to gain exclusive write access to the global accumulators d06a-(run statistics) so that multiple instances of p17 can be run in parallel with each other (see [lgdf-p17.m 197 to 206]).

The process p18-(check for completion) waits until all particle histories are completed, then signals the report-writing process, p14 by "setting" d06a-(run statistics).

Implementation Details

A file is used to specify the target architecture and language (both C and FORTRAN are currently supported). An example

"project" file is shown in Part 1 of Appendix D. The macro-encoded network diagrams for GAMTEB (corresponding to p00, p01, and p02 in Figures 1-3 and also in Part 2 of Appendix D) is shown in "wirelist" form in Part 3 of Appendix D. A data dictionary describing the meanings for global data values is given in Part 4, the FORTRAN data declarations are given in Part 5, and process definitions in Part 6, of Appendix D.

Debugging the Large-Grain Data Flow version

1) To avoid shared variable conflicts (shared memory synchronization problems) between multiple invocations of the subroutines called by multiple copies of p17, all inputs and outputs of those subroutines were removed from COMMON and made into subroutine arguments. (If this were not done, those subroutines would not be re-entrant, and non-determinism would likely result). While manually moving arguments of subroutine TRACK [lgdf-p17.m 228-300] out of COMMON, one result argument (DLS) was inadvertently not added to the parameter list for TRACK.

   This error resulted in an arithmetic exception (floating point overflow). Fortunately, the problem was diagnosed almost immediately, thus avoiding yet another grueling session with dbx (the 4.2 Unix symbolic debugger). This was the only error that prevented the LGDF version from running, although running correctly took a little more work.

At [lgdf-p17.m 228]:

```
SUBROUTINE TRACK(IA,JA,X,Y,Z,U,V,W,CL,CL2,CRAD2)
```

was changed to:

```
SUBROUTINE TRACK(IA,JA,X,Y,Z,U,V,W,CL,CL2,CRAD2,DLS)
```

2) Data flow bubbles p16 and p18 were incorrectly coordinated, causing abnormal termination. The symptom was that output was not being written, indicating that p14 was not waking up. Examination of the "LGDF trace file", which optionally traces all data flow actions during a run, made it very easy to locate the problem. From the file, it was clear that p14 was lacking only the d06a input. This directed suspicion to p18, which is supposed to set d06a. The trace file also showed that p16 and p17 were coordinating their activities correctly. The conclusion was that p18 was not running correctly during its last execution cycle. Examination of the code showed that p18 was off by one on its count of particles. The inequality in:

```
IF (NPC.GT.NPPC) THEN                                    [lgdf-p18.m 5]
```

was changed to:

```
IF (NPC.GE.NPPC) THEN
```

The data flow control now worked correctly, and (incorrect) output was produced.

3) The output produced by sequential LGDF GAMTEB now consisted almost entirely of zeros, and the subsequent debug session pointed to a problem with the random number generation. By checking the code for p16 and p17, an error in variable identifiers on d13 was found. The global variable KERN, used to supply a seed for use in local random number generation had been confused with the local variable KRN. The result was that the variable in LGDF data definition file d13 did not agree with the name used in the actual code.

Several minor corrections were necessary:

a) In [lgdf-d13] the incorrect global variable declaration:

```
INTEGER KRN
```

was changed to:

```
INTEGER KERN
```

b) At [lgdf-p16.m 11] the local name (KRN):

```
KRN = KRN2
```

was changed to:

```
KERN = KRN2
```

c) At [lgdf-p17.m 15], the declaration of the local variable:

```
INTEGER KERN
```

was changed to:

```
INTEGER KRN
```

At this point, a large portion of the output was correct.

4) The next bug fixed was in the LGDF data definition file. The variables in d11 did not agree with the names used in the

actual code.

5) Another problem was that the global variables in d06 [lgdf-d06] were not updated in p17. As a result, the corresponding values in the output were all zero. The correction involved simply adding the updates, for example:

```
IGNCOL=IGNCOL+NCOL                                    [lgdf-p17.m 197]
```

(see also [lgdf-p17.m 198 to 206])

6) The variable ABSORB was not reset to zero in p17. This resulted in a value of ABSORB in the output that was much too large. The cause was easy to find, and the necessary line of code was added:

```
ABSORB=0                                              [lgdf-p17.m 41]
```

7) The resulting program was run on the VAX with 100 particles and produced the same output as the f77 version described in the previous section.

6. PARALLEL LGDF GAMTEB FOR THE HEP UNDER UPX

Several problems arose associated with getting the parallel LGDF version of GAMTEB running under HEP/UPX.

1) The LGDF process p17-(run history for one particle and descendants) needs to get exclusive update access to d06a in

order to add in the statistical contribution of a particle and its daughter particles. Previously, no LGDF facility existed to "give back" exclusive access to either an input or an output in any way other than by clearing or setting. Therefore, two new macros were added to the LGDF process pseudo-functions: unread_ and unwrite_. An example of the use of the unread_ macro can be found in p17 at [lgdf-p17.m 222].

2) The LGDF macros were not generating EXTERNAL declarations for processes that were created. This was initially handled by adding the EXTERNAL declarations manually. Fixing the LGDF macros took about an hour. See for example [lgdf-p00.f 376 to 378].

3) When the self-scheduled LGDF version of GAMTEB was first run in parallel on the HEP (initially at Argonne because the Los Alamos HEP was unavailable), the processes deadlocked after only a few data flow actions. The problem was traced to a mistake in the generated HEP self-scheduling code associated with wake-up barriers for processes with more than one output datapath. The error involved the omission of an LAREAD for asynchronous variables during process wakeup. (The code for HEP-OS was correct, the translation for UPX was in error). The fix was a change to part of one line in the LGDF macros, and took about five minutes. The error was difficult to diagnose, and was found during a visit to Los Alamos using the sgsdb debugging facility[7].

An example of an erroneous LGDF process execution barrier that was produced is:

```
CALL AWRITE($DW(5),GO)
CALL AWRITE($DW(6),$DW(5))
CALL AWRITE($DW(7),$DW(6))
GO=LAREAD($DW(7))
```

which should be:

```
CALL AWRITE($DW(5),GO)                          [lgdf-p10.f 26 to 29]
CALL AWRITE($DW(6),LAREAD($DW(5)))
CALL AWRITE($DW(7),LAREAD($DW(6)))
GO=LAREAD($DW(7))
```

4) Another change was needed because the version of HEP/UPX we used does not automatically interlock parallel attempts at I/O. No error indication is given, the I/O just behaved erratically. As an example, the prompt for user input written to Unit 6 (stdout) would occasionally show up on the terminal, but most often would appear in the trace file (opened as Unit 4)! The macros lockio_ and unlockio_ were added to provide a way for users to do mutual exclusion of I/O between user processes and all other I/O (including trace file writes) in a transparent way. Examples can be seen in p14 at [lgdf-p14.m 13 and 105].

GAMTEB was then run successfully in parallel LGDF mode with 12 parallel processes (including six clones of p17) and produced correct results.

When we attempted to run timing tests with 30 clones of p17, a strange HEP/UPX system error occurred which is still being investigated at this writing. When the problem is resolved, timing curves will be submitted as an added appendix to this report.

Roadblocks to progress:

1) The HEP at LANL was often unavailable during the period of this contract. On several occasions it was down for a week or more (for example, when the new operating system was installed). On another occasion, the HEP appeared to be down when it was not, due to a change in login procedure that we were not aware of.
2) During those times, we attempted to proceed with work on GAMTEB by using the HEP at Argonne. Unfortunately, the Argonne HEP down-time seemed frequently to overlap with that of the LANL HEP. (One reason for this is that the Argonne HEP also installed HEP/UPX during this time period).
3) We frequently had difficulties with remote logins at both Argonne and LANL. This often took a dozen or more tries. It was sometimes rather frustrating to then discover that the HEP was down!
4) On both HEP's, we encountered several serious problems:

    a) Compilation was sometimes random. A source file might produce error messages during several attempts to compile it, then compile successfully (unchanged) at a later time.

b) A problem with the linker made it difficult to get an executable version. This was eventually fixed.
c) The current problem with the disk (described elsewhere) prevents running the benchmarks.

7. CONCLUSIONS AND RECOMMENDATIONS FOR FURTHER WORK

The portability problems (discussed in Section 3), we encountered with getting the original (non-LGDF) GAMTEB running on the VAX (sequentially!) were much more difficult and time-consuming than any of the problems encountered in either creation of the LGDF model version, or in fixing the problems with the generated parallel self-scheduler on the HEP. This was something of a surprise, since we expected the opposite to be the case.

Based on the experiences related in Section 4, if Large-Grain Data Flow techniques are to be applied practically in converting existing large-scale scientific application codes for parallelism, a "smart data flow editor" would be necessary to avoid making the kinds of minor "bookkeeping errors" we encountered.

Future Work

Specific future tasks contemplated related to this work include:

1) Complete speedup experiments for the parallel LGDF version of GAMTEB.

2) Add graphic tracing specifications to the GAMTEB "wirelist" file for graphic execution monitoring.

3) Develop a graphics editor for entering LGDF network diagrams (to run on a variety of inexpensive graphics terminals--initially on IBM PC and Tektronix 4105 class intelligent graphics terminals).

4) Investigate techniques for "non-intrusive" LGDF parallel process execution monitoring. The aim is to provide rapid, real-time visual feedback on run-time characteristics for various parallel algorithms in such a way that the monitoring does not significantly distort the run-time characteristics compared to an unmonitored LGDF program.[*]

5) Develop an LGDF scheduler for CRAY X-MP class parallel processors. This would involve both multi-tasking using the CRAY Multi-tasking Library and "micro-tasking" by generating specialized CRAY assembly language (CAL) scheduling instructions.

6) Develop an LGDF scheduler for "hypercube" architectures without shared memory, such as the Intel iPSC.

Conclusions

During the period of this research work, the HEP underwent a major change in operating system as pre-release versions of the

---

[*] A videotape, "The Traffic Light Demo", showing an example of a color graphic execution monitoring facility for an LGDf program, is available from the authors of this report.

HEP Unix Parallel Executive (UPX) gradually replaced the earlier HEP-OS system. Because of the transparent parallel operation made possible by the LGDF approach (i.e., users do not code any explicit parallel calls or synchronization statements), this shift was accommodated relatively easily and caused only minor problems.

We see the problem of lack of portability of parallel scientific application codes as becoming a major problem, particularly for the National Laboratories. The techniques described in this report show promise for providing a solution to this problem.

The long-term goal of our work with Large-Grain Data Flow is the development of a scientific programming environment that will help bridge the gap between the current sequential FORTRAN environment and both current and future parallel processor supercomputers.

REFERENCES

[1] R. G. Babb II and J. M. Hardy, "Applications of Program/System design techniques to the CRAY-1," paper presented at the 1980 ACM Mountain Region Conference, Denver, Nov. 15, 1980.

[2] J. M. Hardy and R. G. Babb II, "Speedup of FORTRAN subprograms ROTATE and ALAG3D on the CRAY-1 using Program/System methods," Final Report for Optical Systems Branch, Air Force Weapons Laboratory, Contract F2965079, D0028-5019, Summers Engineering, Inc., October 1980.

[3] J. M. Hardy and R. G. Babb II, "A Program/System solution to the EOS (Equation of State) Problem," Final Report for Lawrence Livermore National Laboratory under University P.O. 3196401, Summers Engineering, Inc., May 1981.

[4] R. G. Babb II, "Data-driven implementation of data flow diagrams," in Proc. 6th Int. Conf. on Software Engineering, Tokyo, Japan, Sept. 1982, pp. 309-318.

[5] R. G. Babb II, "Parallel Processing with Large-Grain Data Flow Techniques," Computer, Vol. 17, No. 7, July 1984, pp. 55-61.

[6] R. G. Babb II, "Programming the HEP with Large-Grain Data Flow Techniques", in MIMD Computation: HEP Supercomputer and Its Applications, (ed. by J. S. Kowalik). Cambridge, MA: The MIT Press (to appear May 1985).

[7] D. Carstensen, "How to use 'sgsdb' and other observers", addendum to Proc. Workshop on Parallel Processing using the Heterogeneous Element Processor, Norman, OK, March 1985.

APPENDIX A

Programming the HEP with Large-Grain Data Flow Techniques

(preprint)

3.5 PROGRAMMING THE HEP WITH LARGE-GRAIN DATA FLOW TECHNIQUES†

R. G. BABB II
Department of Computer Science and Engineering
Oregon Graduate Center
Beaverton, Oregon

1. INTRODUCTION

Programming parallel processors can be very frustrating. In addition to the usual software engineering problems common to all forms of program development, an additional set of problems must be avoided and additional criteria must be met for a parallel program to be judged successful. Software engineering problems directly related to the introduction of parallelism include:

- Deadlock and livelock avoidance
- Preventing race conditions
- Avoiding creation of too many parallel processes
- Detecting program termination

New evaluation criteria for parallel programs include:

- Program speedup versus number of processors
- Size of synchronization overhead
- Effect of problem size on speedup
- Max. number of processors that can be kept busy
- Is the program deterministic?

In addition, new software design issues arise, such as:

- What size program "chunks" should be used?
- How many parallel processes should be created?
- What form of process synchronization should be adopted?
- How should access to shared data be managed?
- How can deterministic program execution be guaranteed?

† This work was supported in part by Denelcor, Inc., Aurora, Colorado, under Contract 84-S-010, and Los Alamos National Laboratory under Contract 9-Z-34-P-3915-1.

- How should be processing tasks be sub-divided to make the most effective use of available parallel hardware?

Debugging parallel programs is notoriously difficult. Race conditions can masquerade as program logic errors. When deadlock occurs on the HEP, for example, the addresses where the various parallel processes are "hung" can be determined. However, figuring out how the program got *into* the deadlock situation is usually much more difficult. Debug tracing can affect the parallel behavior of the program being debugged. Non-deterministic programs sometimes "fix themselves" when debug tracing is added, since the tracing serializes a portion of the execution.

The Large-Grain Data Flow (LGDF) methods described in this chapter represent an attempt to provide an abstract computational model for parallel processing that is easy to understand, yet powerful enough to address the questions and issues listed above. Another goal of the methods is to provide a model that can be implemented on a wide variety of *both* parallel and sequential architectures. In this paper the discussion and all of the examples refer to HEP-OS Fortran77[1] for the Denelcor HEP-1 parallel processor[2].

The next section presents an introduction to the LGDF computational model, notation, and semantics. Section 3 presents an overview of the steps involved in using the LGDF Macro Toolset to implement parallel programs. In Section 4 the solution to a small numerical programming example is presented—the parallel solution of a triangular system. Conclusions and references to related work can be found in Section 5.

2. LARGE-GRAIN DATA FLOW

Considerable research effort during the past 15 years has been devoted to the study of dataflow machine architectures and languages as a means to achieve highly parallel computation[3]. The source of parallelism in the data flow approach arises from the possibility of simultaneous execution of a large number of independent operations whose operands have been previously computed. Operations are conceptually linked in a network so that the result of each local computation is fed automatically into the appropriate inputs of other operations. Although traditional dataflow approaches provide an attractive basis for parallel processing, only a few experimental dataflow machines have actually been built[4] [5] and data flow languages such as ID[6], VAL[7], SISAL[8] and LAU[9] have not been widely accepted.

A major parameter in parallel processing is the size of the "granule" of computation that is executed in parallel. In traditional programs for Von Neumann computers, a granule corresponds to an entire application program, and little parallelism within an application program can be exploited. On the other hand, in most dataflow work to date, the grain size chosen for parallel scheduling has been at

the level of a single arithmetic or logical operator. Large-Grain Data Flow combines features of both approaches. The large grain structure of application programs is represented explicitly.

The LGDF computation model resembles traditional data flow in that LGDF processes are activated and controlled by the arrival of and consumption of data values. An *LGDF process* can be in one of three process-states: *executing*, *suspended*, or *terminated*. While executing, in addition to arithmetic and logical computation on input data values it can also perform data flow control actions corresponding to:

- the consumption of data values
- the production of data values
- changing the case-state of process
- reiteration of a process
- suspension of a process

All processes are initially suspended. Processes are represented diagrammatically by circles as shown in Fig. 1. Each process is given an associated descriptive name, and a unique p# tag. An *LGDF data path* is a data memory shared among a number of LGDF processes. Data paths are represented by several types of directed arcs, as shown in Fig. 2. Each data path is also given a descriptive name, and has associated a unique d#. In addition to data values, each data path has a data-state: *empty* or *full*. All data paths are initially empty. Processes and data paths are linked together into acyclic networks of producers and consumers of data as shown in Fig. 3. LGDF networks can achieve parallel operation based on simple producer/consumer data flow interactions. Processes are activated asynchronously depending only upon the empty/full data-states of their associated input and output data paths. All LGDF processes must obey the following two rules:

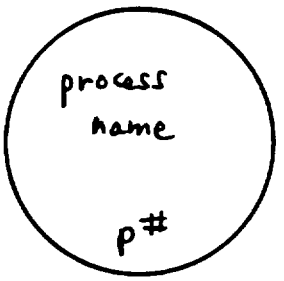
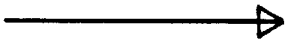


Fig. 1. Graphic representation of an LGDF process.

(a) Clearable data path.



(b) Non-clearable data path.

(c) Side-effect (non-controlling) data path.

Fig. 2. Graphic representation of LGDF data paths.

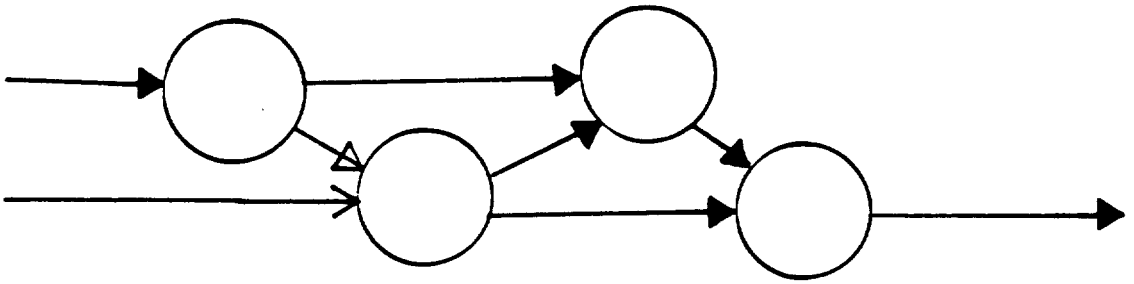


Fig. 3. Graphic representation of an LGDF process network.

Execution Rule. An LGDF process may change its process-state from suspended to executing only after all of its associated input data paths are full, and all of its output data paths are empty.

Data Flow Progress Rule. Upon suspension of an execution cycle, an LGDF process is required to have made data flow progress. This means that it has cleared (consumed) at least one input, or it has set (produced) at least one output during the current execution cycle. Otherwise, the process is terminated.

Setting an output data path has the effect of making the data path available to activate downstream processes. In a similar fashion, processes that clear their input data paths can indirectly activate upstream processes by making a data path available for writing. During any one execution cycle for a process, each of its input data paths can be cleared at most once. Similarly, each of its output data path can be set at most once. Processes are allowed to read data values

only from full input data paths and write data values only on empty output data paths.

Processes can have two types of access to values on input data paths:

- *Read-only*—associated data values may be referenced, but not changed.
- *Update*—associated data values may be both read and changed.

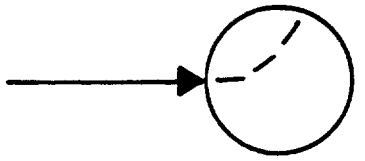
Input data paths that are of type update correspond to variables in ordinary programming languages. Read-only data paths correspond, for example, to constants or call-by-value function arguments. Graphic notations for read-only and update data paths are shown in Fig. 4. LGDF processes are restricted to write access to their output data paths.

Access to data values, for example, arrays, can be shared among a set of LGDF processes in two different ways:

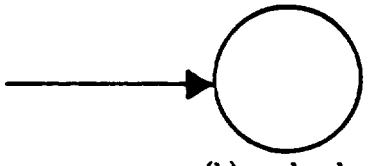
- sequential shared access
- parallel shared access

A sequentially shared data path represents a data memory which is controlled so that at most one process has access to the shared data values at a time. This corresponds to forcing serialization of that portion of the computation. A parallel shared data path represents a data memory in which asynchronous access is possible by a set of processes. Processes that access a parallel shared data path can also compete non-deterministically for exclusive access to the shared input or output. Notations for sequential and parallel shared data paths, for both read-only and update access, are shown in Fig. 5.

LGDF process networks can also be defined hierarchically. This means that any node in a network may be specified either by an LGDF process (as discussed above) or by another network. The semantics of this are the same as if all references to lower level networks were replaced by their defining networks. This

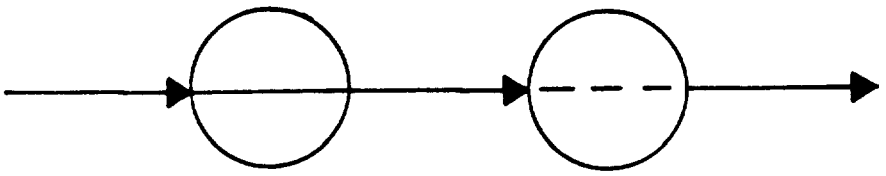


(a) update

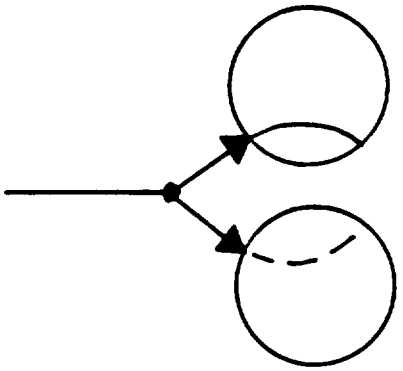


(b) read-only

Fig. 4. Graphic representations for update and read-only data paths.



(a) Sequential shared read-only and update data paths.



(b) Parallel (asynchronously) shared read-only and update data paths.

Fig. 5. Graphic notations for shared data paths.

grouping mechanism is used to cluster processes that execute with approximately the same frequency[1].

LGDF application designs tend to deal with data-activated chunks that correspond typically to 5 to 50 lines of executable higher-level language statements. This has the effect of providing a relatively familiar subroutine-like interface for programmers, as shown in Fig. 6.

LGDF network diagrams specify unambiguously which SUBROUTINE parameters are inputs (X), which are outputs (Y,J) , and which are both (I). Another difference is that a programmer does not CALL a SUBROUTINE for execution, but activates it indirectly by sending data to it.

[1]On a single PEM HEP, the process grouping information is not used directly, since all processes are effectively at the same scheduling level (a 128-way single-level parallel process scheduler is built in to the hardware). However, when emulating LGDF parallel operation on a sequential computer, the round-robin scheduling mechanism can be made more efficient, because processes that execute infrequently are checked less often for executability.

SUBROUTINE A(X,I,Y,J)

.

.

.

RETURN
END

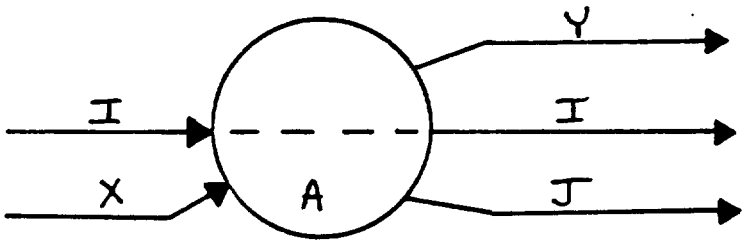


Fig. 6. Comparison between a SUBROUTINE and an LGDF process.

3. USING LGDF MACROS FOR PARALLEL PROGRAMMING

In this section, we give an overview of the steps involved in implementing a parallel Fortran program on the HEP using the Large-Grain Data Flow Macro Toolset. The steps involved in using the LGDF macro tools are:

1) *Draw LGDF Network Diagrams*—The goal in designing an LGDF process network to solve a problem is to produce a hierarchical, consistent[2] set of network data flow graphs that embody the large grain logical data dependencies inherent in the problem.

   Parallelism can be achieved in several ways. The simplest kind of parallelism in the LGDF computation model is *data-independent* parallel processing. Any two executable LGDF processes that share no data paths can safely execute simultaneously since they cannot interfere with each other.

   The next simplest method is *data-sequential* or *pipelined* parallelism. In a pipeline, a series of processes can be active simultaneously on different phases of the production of a final result. This is the same idea exploited in the floating point units of vector processor supercomputers and in UNIX[3] pipes[10].

   The third method is *asynchronous-parallel-update.* This is "risky" parallel processing, where multiple processes have asynchronous read and/or update access to a shared data structure (usually an array). It is the LGDF programmer's responsibility to ensure that the updates are performed safely and correctly. A common way to ensure this safety is to prevent two processes from trying to update the same array element at the

[2]A set of data flow graphs is consistent if the inputs and outputs of each LGDF system process match the input and output data paths shown on the corresponding lower level defining network.

[3]UNIX is a trademark of Bell Laboratories.

same time.

The Triangular Solver example contains examples of all three types of parallelism.

2) *Create Wirelist File*—The Wirelist file uses macros to declare names and internal tags for processes and data paths. The data flow dependencies of the set of data flow diagrams from step 1) are also encoded using macro calls[4]. The macros indicate which data paths are inputs and which are outputs for each process and process network. Also encoded are the type of access each process has to its input data paths: clearable (CL) or non-clearable (NC), and read-only (RO) or update (UP).

3) *Package Data Declarations*—Fortran data declarations corresponding to each data path in the data flow diagrams are put into separate files whose names correspond to their data path tags. (These become labeled COMMON declarations in the generated programs).

4) *Write LGDF Programs*—Combine LGDF macro calls with appropriate Fortran code to implement each program. The LGDF programmer writes macros for execution barriers and actions. Data flow control macros are used to signal consumption of inputs and production of output values. Other macros are used to change the case-state of a process, and to reiterate, or suspend execution.

5) *Macro Expand LGDF Programs*—The wirelist and data declaration files are used to control the macro-expansion of the LGDF program files to produce compilable Fortran for a particular machine. The expansion is based on the data path connectivity information encoded in the Wirelist File. SUBROUTINE headers and labeled COMMON statements corresponding to a program's input and output data paths are inserted automatically.

6) *Compile and Execute*—The resulting source code is then compiled including, if desired, pre- and/or post-compilation optimization steps available for the particular target environment.

See [11] and [12] for further details on LGDF methods for software engineering and parallel processing.

---

[4]This step is currently done manually, but will eventually be an automatic result of a graphics-based tool used to draw LGDF network diagrams.

4. SOLVING A TRIANGULAR SYSTEM

Presented below is a detailed example of LGDF programming for the HEP. The development of the example assumes familiarity with the terms and notation defined in Section 2. All user-coded inputs to the macro expansion process are shown, as well as samples of the expanded Fortran code.

The problem is to solve a lower triangular matrix T (of dimension N by N) to yield a result vector Y (of dimension N). The parallel solution strategy employed is to break T up into sub-matrices of dimension (usually) K by K. Since K may not evenly divide N, we will in general have a column block of submatrices left over that will be less than K columns wide. We choose to put this narrow column of submatrices at the left edge of T, as shown in Fig. 7 for the case N=8, K=3.

Again, referring to Fig. 7., the basic approach is employ a triangle solver (TS) to update Y for the first triangular group of T values labeled TS_1. After this TS process has finished updating the result vector Y, a series of matrix multiply (MM) processes can begin execution in parallel for the column block below. Each of the matrix multiply steps is independent. However, care must be taken that parallel updates to the result vector Y by the various MM's are performed safely.

The next triangle solver step on the diagonal (TS_2) must wait until all of the matrix multiply steps in the same row block to its left have completed their updates of Y. Then a series of matrix multiply processes can be started in the column block below, and the pattern repeats. The LGDF solution to this problem employs an "interlock" data path to block the next triangle solver step until

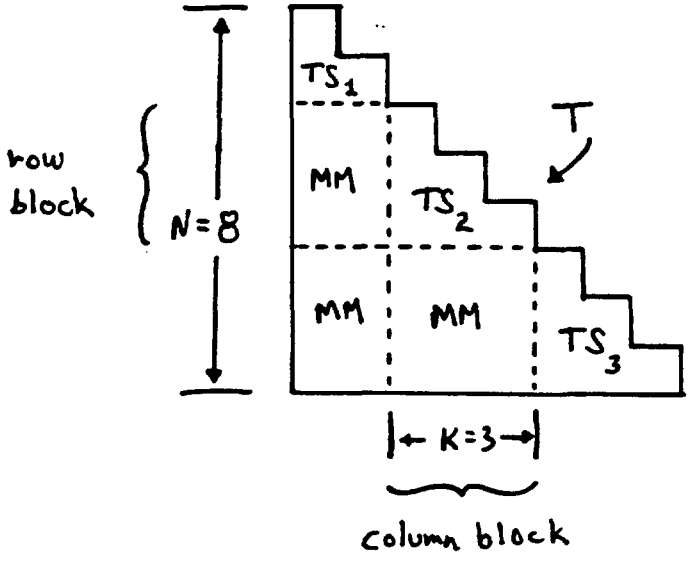


Fig. 7. Solution strategy for the parallel triangular solver.

the appropriate matrix multiply steps have completed. The interlock is cleared by use of a matrix multiply "check-in" process (MMCKIN) that counts the number of matrix multiplies that have completed in each row block. When the row block of MM's has finished, a row block completion signal is produced that causes the triangle solver interlock data path to be cleared.

In the discussion of the LGDF solution below, p and d numbers are used to cross-reference the various networks, processes, and data paths with program texts.

4.1. *Draw LGDF Network Diagrams*

The LGDF solution to the parallel triangular matrix solver is shown on three process network diagrams, in Figs. 8-10. The top level network (p00), shown in Fig. 8, consists of a process TSINIT (p10) that sets up a problem to be solved (d01), initializes the result vector Y (d02) to zeros, and initializes various counters and index values (d03, d04). TSINIT can execute immediatedly because all internal data paths in a network hierarchy are initially empty. The system[5] TMWORK (p01) then updates the result vector Y in place to produce the final result vector (d02a)[6]. The dashed lines inside the circle for p01 for data paths d03 and d04 indicate that their contents can be updated by processes running inside p01. Note that data path d01 can be cleared by processes internal to p01, but the associated TS problem values can not be changed.

An LGDF process network definition of TMWORK (p01) is shown in Fig. 9. The notation in square brackets "[p00]" below p01 on Fig. 9 indicates the parent (context) network process for a network. The program TSWORK (p11) controls all of the remaining processes in the solution, either directly or indirectly. Since it "knows" the global state of computational progress, it can produce the answer by setting the result vector Y (d02a) after all final Y values have been computed. TSWORK causes a triangle to be solved by TS (p12) by setting appropriate TS control values on d04a. It then blocks itself from further execution until the appropriate matrix multiplies have completed inside MMWORK (p02) by setting the next row block interlock NRBI (d05). NRBI is cleared by WAITRB (p13) after the next row block complete signal NRBC (d07) is produced inside p02. After TS has updated Y (d02), it starts the matrix multiplies below it by setting appropriate MM control parameters (d06). Note that both p12 and the processes inside p02 have asynchronous update access to Y (d02), a potentially unsafe situation.

---

[5]We use the term "system" for processes that are defined by a network, rather than by an LGDF program.

[6]Note that even though d02 and d02a represent the same array, they are assigned different data path tags, and can be independently set and cleared. The open arrowhead on data path d02 indicates that no process inside p01 is allowed to explicitly clear the result vector. (It is said to be *non-clearable*). However, a mechanism is provided in the generated Fortran code to automatically propagate a "clear" backwards for shared data paths when the corresponding shared output is cleared, which in this case would have to be done by a process external to this network.

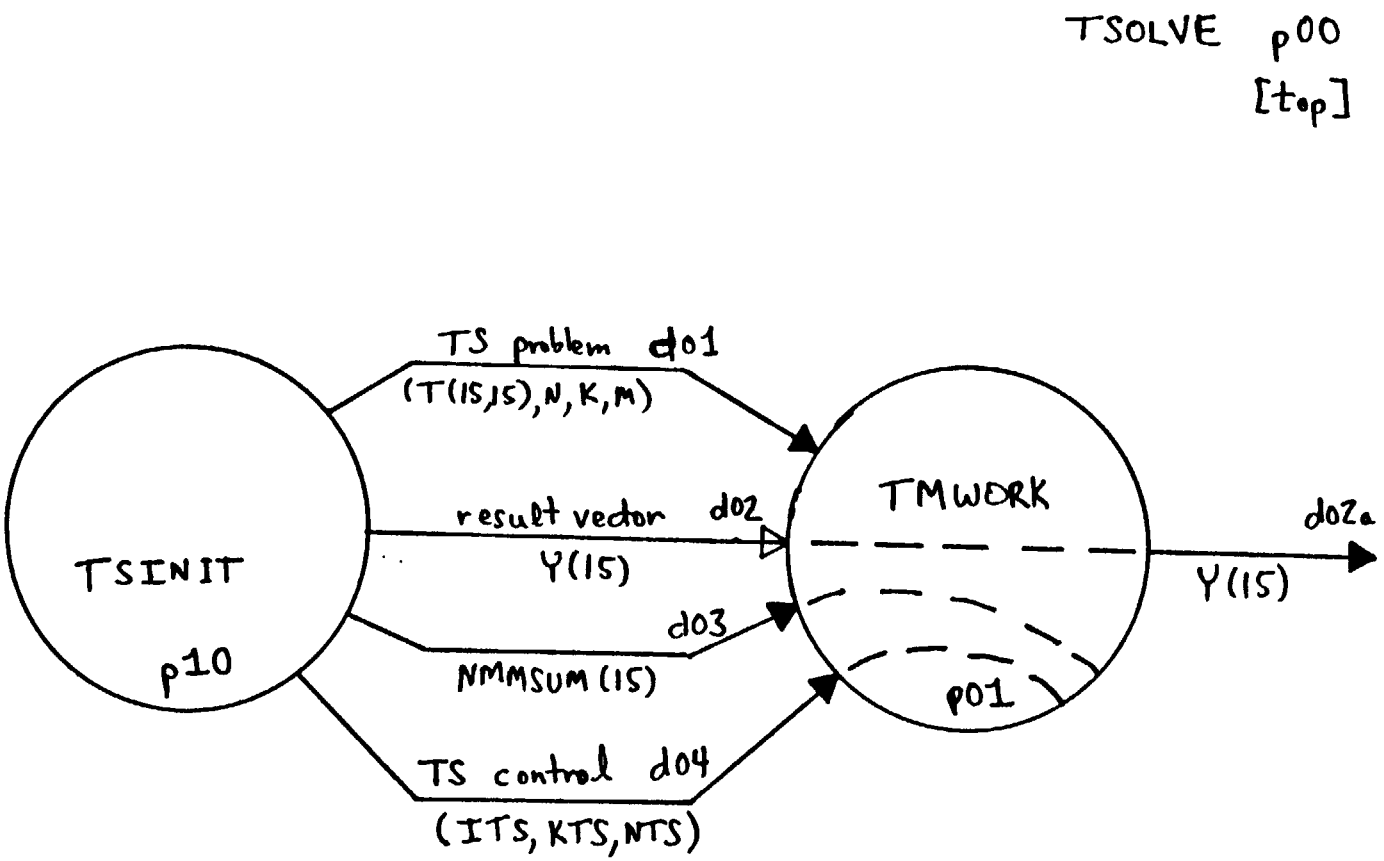


Fig. 8. LGDF process network for TSOLVE (p00).

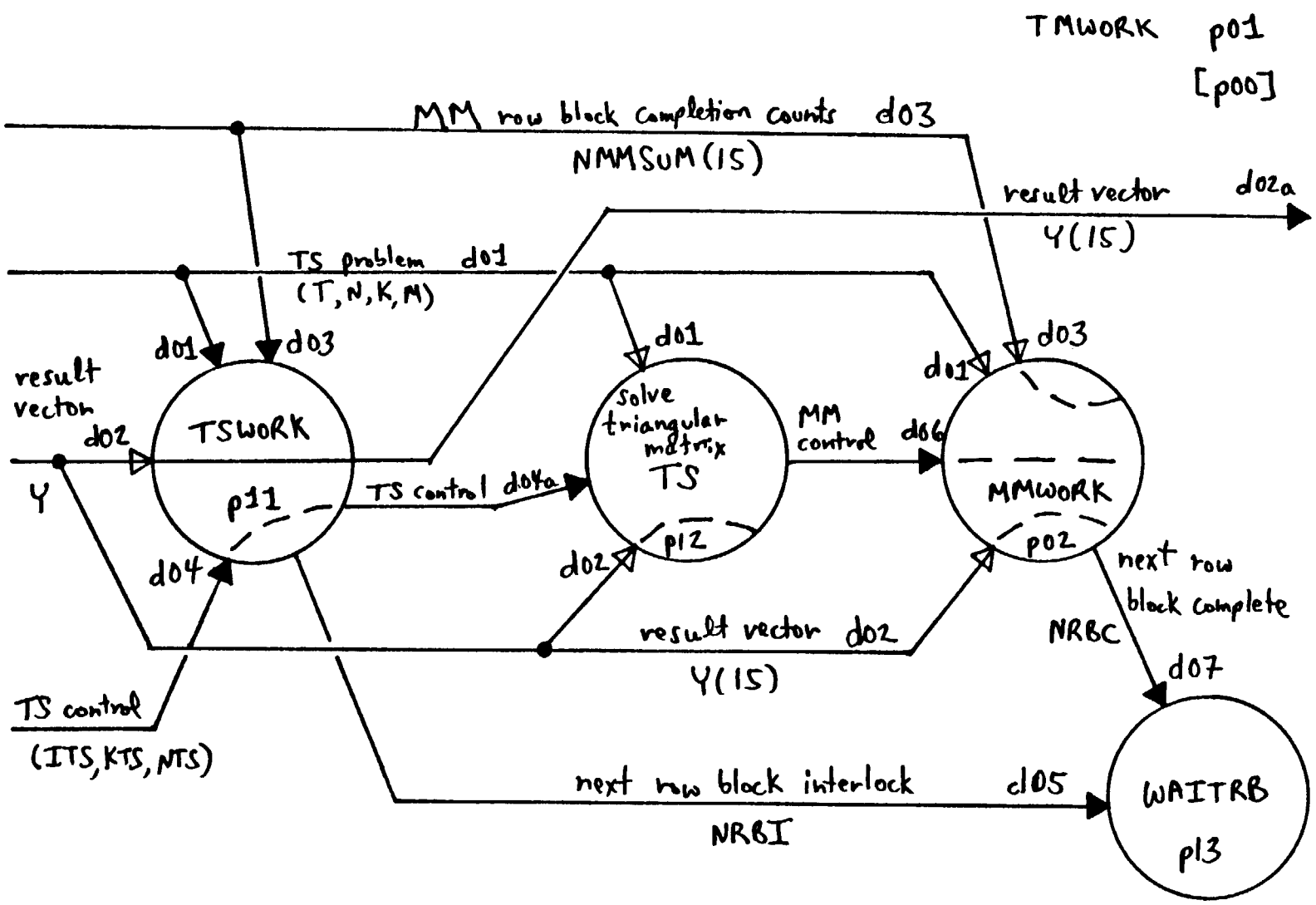


Fig. 9. LGDF process network for TMWORK (p01).

The lowest level LGDF process network for this example is for MMWORK (p02), shown in Fig. 10. MMWORK (p14) generates the next MM sub-problem (d06a). A collection of MM processes (p15) compete asynchronously for work on d06a. After an MM process gets exclusive read access to d06a, it makes local copies of the MM control values and clears d06a, allowing p14 to generate the next MM sub-problem immediately. Each MM computes its contribution to the result vector Y using an array local to each copy of p15. When it has completed its computation, it competes for exclusive write access to the MM checkin data path (d08). Since only one MM process can be checking in at a time, it can also safely update Y during this time. MMCKIN (p16) counts the number of MM processes that have completed by row, and when all have checked in, sets the next row block completion signal NRBC (d07).

4.2. *Create Wirelist File*

A *Wirelist File* is a set of macro calls consisting of two parts. The first part, shown in Fig. 11, defines data path and program names and tags. The tags are then used in the second part, which consists of a series of macro calls that encode the connectivity, and data path and process types of a set of LGDF process network diagrams. The wirelist corresponding to the diagrams in Figs. 8-10 are shown in Fig. 12. Error checking is incorporated in the macro expansion process. Errors diagnosed include use of an undefined d or p tag and input of a data path that is neither an external input, nor the output of another LGDF program. Data path type errors are also diagnosed, such as inconsistent usage of clearable and non-clearable data paths.

4.3. *Package Data Declarations*

Data declarations corresponding to data paths are retrieved from separate files. The data declaration files for this example are shown in Fig. 13.

4.4. *Write LGDF Programs*

The complete LGDF macro forms for programs p10 through p16 are shown in Fig. 14.

4.5. *Macro Expand LGDF Programs*

A sample program (p12) macro-expanded for the HEP is shown in Fig. 15. Also generated is a top-level initiation program and a set of network initiator subroutines, one for each network diagram. The top-level initiation program that contains trace aid facilities and code which prints out run statistics and the final set of data path states. It also calls the top-level network initiator subroutine (in this case p00) which creates the other LGDF processes in the network. Each process can be created only once in the current version of the tools. Network initiator subroutines for TSOLVE (p00), TMWORK (p01), and MMWORK (p02) are shown in Fig. 16.

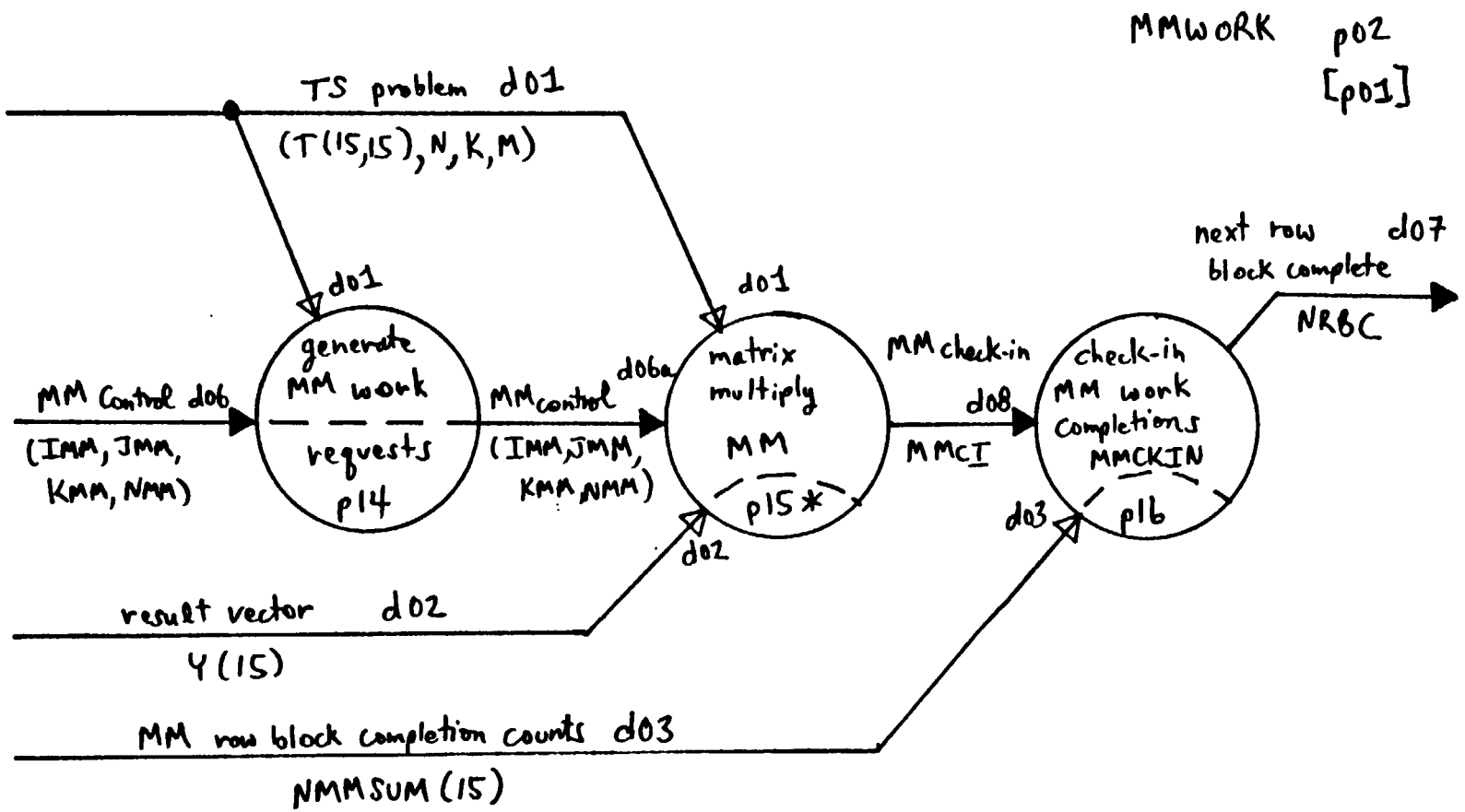


Fig. 10. LGDF process network for MMWORK (p02).

```
*** DEFINE DATA PATHS

***     tag  short name     descriptive name

_defdp(d01, "TS problem", "(T(15,15),N,K,M)")
_defdp(d02, "Y",          "result vector (space)")
_defdp(d03, "NMMSUM",     "MM row block completion counts")
_defdp(d04, "TS control", "(ITS,KTS,NTS)")
_defdp(d05, "NRBI",       "next row block interlock")
_defdp(d06, "MM control", "(IMM,JMM,KMM,NMM)")
_defdp(d07, "NRBC",       "next row block complete")
_defdp(d08, "MMCI",       "MM check-in")

*** DEFINE PROGRAMS & NETWORKS

***     tag  short name     descriptive name

_defpn(p00, "TSOLVE",     "parallel triangular matrix solver")
_defpn(p01, "TMWORK",     "perform TS and MM work")
_defpn(p02, "MMWORK",     "perform MM work")
_defpn(p10, "TSINIT",     "initialize TS test problem parameters")
_defpn(p11, "TSWORK",     "generate TS work requests")
_defpn(p12, "TS",         "solve triangular matrix")
_defpn(p13, "NEXTTS",     "interlock TS and MM work progress")
_defpn(p14, "MMWORK",     "generate MM work requests")
_defpn(p15, "MM",         "matrix multiply")
_defpn(p16, "MMCKIN",     "checkin MM work completions")
```

Fig. 11. Data path and program definitions for the parallel triangular solver example.

```
*** WIRELIST

*****************************
_net(top, [])
   _sys(p00, [top])
      _waitout(d02a,CL)
   _endsys(p00, [top])
_endnet(top, [])
*****************************
_net(p00, [top])
   _prog(p10, [p00])
      _out(d04,CL)
      _out(d01,CL)
      _out(d02,NC)
      _out(d03,CL)
   _endprog(p10, [p00])
   _sys(p01, [p00])
      _in(d01,CL,R0)
      _in(d02,NC,UP)
      _in(d03,CL,UP)
      _in(d04,CL,UP)
         _shared(d02,d02a,NC,UP)
      _out(d02a,CL)
   _endsys(p01, [p00])
_endnet(p00, [top])
*****************************
_net(p01, [p00])
   _in(d01,CL,R0)
   _in(d02,NC,UP)
   _in(d03,CL,UP)
   _in(d04,CL,UP)
   _prog(p11, [p01])
      _in(d01,CL,R0)
      _in(d02,NC,R0)
      _in(d03,CL,R0)
      _in(d04,CL,UP)
         _shared(d02,d02a,NC,R0)
         _shared(d04,d04a,CL,UP)
      _out(d04a,CL)
      _out(d02a,CL)
      _out(d05,CL)
   _endprog(p11, [p01])
   _prog(p12, [p01])
      _in(d01,NC,R0)
      _in(d02,NC,UP)
      _in(d04a,CL,R0)
      _out(d06,CL)
   _endprog(p12, [p01])
   _sys(p02, [p01])
      _in(d01,NC,R0)
      _in(d06,CL,R0)
      _in(d02,NC,UP)
      _in(d03,NC,UP)
      _out(d07,CL)
   _endsys(p02, [p01])
   _prog(p13, [p01])
      _in(d07,CL,R0)
      _in(d05,CL,R0)
   _endprog(p13, [p01])
_endnet(p01, [p00])
*****************************
_net(p02, [p01])
   _in(d01,NC,R0)
   _in(d06,CL,UP)
   _in(d02,NC,UP)
   _in(d03,NC,UP)
   _prog(p14, [p02])
      _in(d01,NC,R0)
      _in(d06,CL)
         _shared(d06,d06a,CL,UP)
      _out(d06a,CL)
   _endprog(p14, [p02])
   _prog(p15, [p02], *6)
      _in(d01,NC,R0)
      _in(d06a,CL,R0)
      _in(d02,NC,UP)
      _out(d08,CL)
   _endprog(p15, [p02])
   _prog(p16, [p02])
      _in(d08,CL,R0)
      _in(d03,NC,UP)
      _out(d07,CL)
   _endprog(p16, [p02])
_endnet(p02, [p01])
*****************************
```

Fig. 12. Encoded data path/program connectivity in the Wirelist File.

```
d01:

T,N,K,M
      REAL T(15,15)
      INTEGER N,K,M
```

```
d02:

Y
      REAL Y(15)
```

```
d03:

NMMSUM
      INTEGER NMMSUM(15)
```

```
d04:

ITS,KTS,NTS
      INTEGER ITS,KTS,NTS
```

```
d05:

NRBI
      INTEGER NRBI
```

```
d06:

IMM,JMM,KMM,NMM
      INTEGER IMM,JMM,KMM,NMM
```

```
d07:

NRBC
      INTEGER NRBC
```

```
d08:

MMCI
      INTEGER MMCI
```

Fig. 13. Data declaration files for the parallel triangular solver example.

```
program_(p10,[p00])
C--LOCAL VARIABLES
      REAL TEMP
      INTEGER I,J
begin_
C--SET UP TEST PROBLEM VALUES (N IS ASSUMED TO BE .GE. K)
      N=8
      K=3
      M=((N-1)/K)+1
C--INITIALIZE THE RESULT VECTOR Y AND T
      DO 5 I=1,N
         Y(I)=0.0
    5 CONTINUE
      DO 20 J=1,N
         DO 10 I=J,N
            TEMP=I*J
            T(I,J)=TEMP
            Y(I)=Y(I)+TEMP
   10    CONTINUE
   20 CONTINUE
      set_(d02)
      set_(d01)
C--INITIALIZE THE ROW BLOCK COMPLETION COUNTS
      DO 50 I=1,K
         NMMSUM(I)=0
   50 CONTINUE
      set_(d03)
C--INITIALIZE TS WORK VARIABLES
      ITS=1
      KTS=N-((M-1)*K)
      NTS=1
      set_(d04)
end_(p10)
```

Fig. 14(a). An LGDF program for TSINIT (p10).

```
program_(p11,[pO1])
begin_
states_(sOO,sO1,sO2)
state_(sOO,issue first TS work request)
      set_(dO4)
C--(NOTE: NO SET OF dO5 ROW BLOCK INTERLOCK)
C--CHECK FOR LAST TRIANGLE
      IF(NTS.EQ.M) THEN
         next_(sO2,set result vector)
      ELSE
         next_(sO1,issue next TS work)
      ENDIF
      suspend_
state_(sO1,issue next TS work)
      ITS=ITS+KTS
      KTS=K
      set_(dO4)
C--SET ROW BLOCK INTERLOCK
      NRBI=NTS
      set_(dO5)
C--CHECK FOR LAST TRIANGLE
      NTS=NTS+1
      IF(NTS.EQ.M) THEN
         next_(sO2,set result vector)
      ENDIF
      suspend_
state_(sO2,set result vector)
      set_(dO2)
      WRITE(6,40) (Y(I),I=1,N)
   40 FORMAT(' Y=',15F5.2)
      clear_(dO1)
      clear_(dO3)
      clear_(dO4)
      suspend_
end_(p11)
```

Fig. 14(b). An LGDF program for TSWORK (p11).

```
program_(p12,[pO1])
      INTEGER I,J,LL
begin_
      LL=KTS
      DO 20 J=ITS,ITS+LL-1
         Y(J)=Y(J)/T(J,J)
         LL=LL-1
         DO 10 I=J+1,J+LL
            Y(I)=Y(I)-T(I,J)*Y(J)
   10    CONTINUE
   20 CONTINUE
      clear_(dO4)
      IF(NTS.LT.M) THEN
C -- ENABLE MMWORK
         IMM=ITS+KTS
         JMM=ITS
         KMM=KTS
         NMM=NTS+1
         set_(dO6)
      ENDIF
      suspend_
end_(p12)
```

Fig. 14(c). An LGDF program for TS (p12).

```
program_(p13, [p01])
begin_
      clear_(d05)
      clear_(d07)
      suspend_
end_(p13)
```

Fig. 14(d). An LGDF program for WAITRB (p13).

```
program_(p14, [p02])
begin_
states_(s00,s01)
state_(s00,issue first MM work request)
      set_(d06)
      next_(s01, issue next work request)
      suspend_
state_(s01,issue next work request)
      IMM=IMM+K
      IF(IMM.GT.N) THEN
         next_(s00,issue first MM work request)
         clear_(d06)
      ELSE
         NMM=NMM+1
         set_(d06)
      ENDIF
      suspend_
end_(p14)
```

Fig. 14(e). An LGDF program for MMWORK (p14).

```
program_(p15,[pO2],*)
C--LOCAL VARIABLES
      REAL YHAT(15),TEMP
      INTEGER LI,LJ,LK,LN,I,J,II
begin_
C--WAIT FOR EXCLUSIVE READ ACCESS TO INPUT ARGS
      aread_(dO6)
C--MAKE LOCAL COPIES OF INPUT ARGS
      LI=IMM
      LJ=JMM
      LK=KMM
      LN=NMM
C--ALLOW GENERATION OF MORE MM WORK
      aclear_(dO6)
C--YHAT IS A LOCAL ARRAY USED LATER TO UPDATE Y
      DO 10 I=1,K
         YHAT(I)=0.0
   10 CONTINUE
C
      DO 20 J=LJ,LJ+LK-1
         TEMP=Y(J)
         II=LI
         DO 15 I=1,K
            YHAT(I)=YHAT(I)+T(II,J)*TEMP
            II=II+1
   15    CONTINUE
   20 CONTINUE
C--WAIT FOR EXCLUSIVE WRITE ACCESS TO OUTPUT DATA PATH dO8
C--(ALSO USED TO SERIALIZE UPDATES TO Y BY MM'S)
      awrite_(dO8)
C--UPDATE RESULT VECTOR
      DO 30 L=1,K
         Y(LI)=Y(LI)-YHAT(L)
         LI=LI+1
   30 CONTINUE
C--SIGNAL MM COMPLETION (GIVING CURRENT ROW BLOCK NO. LN)
      MMCI=LN
      aset_(dO8)
      suspend_
end_(p15)
```

Fig. 14(f). An LGDF program for TSINIT (p15).

```
program_(p16,[pO2])
C--LOCAL VARIABLE
      INTEGER LNRB
begin_
C--MAKE LOCAL COPY OF INPUT ARG
      LNRB=MMCI
      clear_(dO8)
      NMMSUM(LNRB)=NMMSUM(LNRB)+1
C -- CHECK FOR ALL MM's CHECKED IN THIS ROW BLOCK
      IF( (NMMSUM(LNRB)+1) .EQ. LNRB) THEN
         NRBC=LNRB
         set_(dO7)
      ENDIF
      suspend_
end_(p16)
```

Fig. 14(g). An LGDF program for MMCKIN (p16).

```
C---- TS -- solve triangular matrix
      SUBROUTINE P12(IPN,IPCTXT)
C-------------- (HEP FORTRAN SYSTEM TABLES) ---------------
      COMMON /SYSTAB/ $LTR,$DW,$DR,LPR(16),LPS(16),LPX(16),LNL(16)
      LOGICAL $LTR,$DW(11),$DR(11)
      LOGICAL GO,DFPROG
C-> d01: TS problem - (T(15,15),N,K,M)
      COMMON /D01/ T,N,K,M
      REAL T(15,15)
      INTEGER N,K,M
C-> d02: Y          - result vector (space)
      COMMON /D02/ Y
      REAL Y(15)
C-> d04: TS control - (ITS,KTS,NTS)
      COMMON /D04/ ITS,KTS,NTS
      INTEGER ITS,KTS,NTS
C-> d06: MM control - (IMM,JMM,KMM,NMM)
      COMMON /D06/ IMM,JMM,KMM,NMM
      INTEGER IMM,JMM,KMM,NMM
      INTEGER I,J,LL
      IF(LPR(IPN).EQ. -1) RETURN
 7799 DFPROG=.FALSE.
      $DR(3)=$DR(3)
      $DR(4)=$DR(4)
      $DR(6)=$DR(6)
      $DW(8)=GO
      GO=$DW(8)
 7798 LPX(IPN)=LPX(IPN)+1
      LPR(IPN)=1
      LPR(IPCTXT)=LPR(IPCTXT)+1
      LL=KTS
      DO 20 J=ITS,ITS+LL-1
         Y(J)=Y(J)/T(J,J)
         LL=LL-1
         DO 10 I=J+1,J+LL
            Y(I)=Y(I)-T(I,J)*Y(J)
   10    CONTINUE
   20 CONTINUE
      GO=$DR(6)
      GO=$DW(6)
      DFPROG=.TRUE.
      IF(NTS.LT.M) THEN
C -- ENABLE MMWORK
         IMM=ITS+KTS
         JMM=ITS
         KMM=KTS
         NMM=NTS+1
         $DW(8)=.FALSE.
         DFPROG=.TRUE.
         $DR(8)=.FALSE.
         DFPROG=.TRUE.
      ENDIF
      GO TO  7797
 7797 IF(.NOT.DFPROG) THEN
         GO=$LTR
         CALL PTRACE(IPN,48,0,LPX(IPN),0.0)
         $LTR=VALUE($LTR)
         LPR(IPN)= -1
         RETURN
      ENDIF
      GO TO  7799
      END
```

Fig. 15. Macro-expanded version of TS (p12) for the HEP.

```
C
C--- HEP FORTRAN PROCESS INITIATORS ---
C
      SUBROUTINE POO(IPN,IPCTXT)
C
C--------------- (HEP FORTRAN SYSTEM TABLES) ---------------
      COMMON /SYSTAB/ $LTR,$DW,$DR,LPR(16),LPS(16),LPX(16),LNL(16)
      LOGICAL $LTR,$DW(11),$DR(11)
 7798 LPX(IPN)=LPX(IPN)+1
      CREATE P10(3,IPN)
        CALL PO1(4,IPN)
      RETURN
      END
C
      SUBROUTINE PO1(IPN,IPCTXT)
C
C--------------- (HEP FORTRAN SYSTEM TABLES) ---------------
      COMMON /SYSTAB/ $LTR,$DW,$DR,LPR(16),LPS(16),LPX(16),LNL(16)
      LOGICAL $LTR,$DW(11),$DR(11)
 7798 LPX(IPN)=LPX(IPN)+1
      CREATE P11(5,IPN)
      CREATE P12(6,IPN)
        CALL PO2(7,IPN)
      CREATE P13(8,IPN)
      RETURN
      END
C
      SUBROUTINE PO2(IPN,IPCTXT)
C
C--------------- (HEP FORTRAN SYSTEM TABLES) ---------------
      COMMON /SYSTAB/ $LTR,$DW,$DR,LPR(16),LPS(16),LPX(16),LNL(16)
      LOGICAL $LTR,$DW(11),$DR(11)
 7798 LPX(IPN)=LPX(IPN)+1
      CREATE P14(9,IPN)
      CREATE P15(10,IPN)
      CREATE P15(11,IPN)
      CREATE P15(12,IPN)
      CREATE P15(13,IPN)
      CREATE P15(14,IPN)
      CREATE P15(15,IPN)
      CREATE P16(16,IPN)
      RETURN
      END
```

Fig. 16. Network initiator subroutines for the parallel triangular solver example.

4.6. *Compile and Execute*

The output of the macro-expansion step is compilable Fortran for the particular target machine. The code can be further pre-processed in environments where Fortran pre-processing tools such as vectorizers are available. Of course, the object code can be further optimized automatically also.

5. CONCLUSION

The basic asynchronous variable mechanism available on the HEP appears deceptively simple. This apparent simplicity can lead even very good programmers, with a good understanding of parallel processing and of the HEP architecture, into very deep asynchronous-parallel update trouble. Several other researchers[13] [14] [15] have concluded that a good way to avoid some of the pitfalls of bare-knuckled parallel processing on the HEP is to program using a relatively simple set of macros to express barriers and other synchronization abstractions. The macros are then expanded into appropriate primitive synchronization actions. Despite the fact that the parallel abstractions supported by the various macro packages are quite different, most people who have tried any of them have found parallel programming much easier. This would appear to be a time to experiment with various abstractions, and to gain experience with software engineering methods to aid in parallel programming. Eventually, the more successful of these abstractions could be encapsulated in later versions of Fortran and other scientific programming languages.

We have come to some unexpected conclusions as a result of our work in modeling algorithms using "safe" parallel (data-independent), pipelined (data-sequenced), and "risky" parallel (asynchronous shared update) LGDF network data flow patterns. The most surprising is that "sequential" programs are often easier to design and implement reliably when based on a parallel (asynchronous) model of computation. The extra difficulty of the traditional implementation approach arises because of the artificial sequencing imposed by conventional programming.

Using the prototype toolset based on macro expansion techniques, we have demonstrated that the the same "macro" program can be run efficiently (unchanged) on a sequential and a parallel processor. The program ran with simulated parallelism on the sequential processor (a VAX) and with real parallelism on the parallel processor (a HEP-1). It is hoped that the software engineering methods described in this section will form the basis for programming a wide variety of parallel architectures.

6. Acknowledgment

The ideas in this paper have been developed over a period of several years, and many people have contributed to their current form. The author would like to acknowledge especially the contributions of James Hardy, Robert Hiromoto, Richard Kieburtz, Richard Hamlet and Danny Sorensen. Dr. Sorensen suggested the Parallel Triangle Solver problem and solution approach, and helped develop both the LGDF process network diagrams and the LGDF programs.

7. REFERENCES

[1] *HEP FORTRAN 77 User's Guide.* Aurora, CO: Denelcor, Inc., 1982

[2] M.C. Gilliland, B. J. Smith, and W. Calvert, "HEP--A semaphore-synchronized multiprocessor with central control," in *Proc. 1976 Summer Computer Simulation Conf.,* Washington, D.C, July 1976, pp. 57-62.

[3] T. Agerwala and Arvind, (eds.), Special Issue of *Computer* on Data Flow Systems, Vol. 15, No. 2, Feb. 1982.

[4] M. Amamiya, et al., "Data Flow Machine Architecture". Internal Notes 1 and 2, Musashino Electrical Communications Laboratories, Nippon Telephone and Telegraph Corporation, Tokyo, Japan, May 1980.

[5] I. Watson and J. Gurd, "A practical data flow computer," *Computer*, Vol. 15, No. 2, Feb. 1982, pp. 51-57.

[6] Arvind, K. P. Gostelow, and W. E. Plouffe, "An Asynchronous Programming Language and Computing Machine," Dept. of Information and Computer Science Report TR114a, University of California, Irvine, CA, Dec. 1978.

[7] J. B. Dennis, "First Version of a Data Flow Procedure Language," in *Lecture Notes in Computer Science*, Vol. 19, Springer-Verlag, 1974, pp. 362-376.

[8] J. R. McGraw, et al., "SISAL: Streams and Iterations in a Single-Assignment Language, Language Reference Manual, Version 1.1", U. of California, Livermore Livermore National Laboratory, Technical Report M-146, July 1983.

[9] D. Conte, N. Hifdi, and J. C. Syre, "The Data Driven LAU Multiprocessor System: Results and Perspectives," in *Proc. IFIP Congress*, 1980.

[10] B. W. Kernighan and R. Pike, *The UNIX Programming Environment.* Englewood Cliffs, NJ: Prentice-Hall, Inc., 1984.

[11] R. G. Babb II, "Data-driven implementation of data flow diagrams," in *Proc. 6th Int. Conf. on Software Engineering*, Tokyo, Japan, Sept. 1982, pp. 309-318.

[12] R. G. Babb II, "Parallel Processing with Large-Grain Data Flow Techniques," *Computer*, Vol. 17, No. 7, July 1984, pp. 55-61.

[13] H. F. Jordan, "Structuring Parallel Algorithms in an MIMD, Shared Memory Environment," to appear in *Proc. 18th Hawaii Int. Conf. on System Sciences,* Jan. 1985.

[14] E. L. Lusk and R. A. Overbeek, "An Approach to Programming Multiprocessing Algorithms on the Denelcor HEP," Argonne National Laboratory, Mathematics and Computer Science Division, Technical Report No. ANL-83-96, Dec. 1983.

[15] E. L. Lusk and R. A. Overbeek, "Implementation of Monitors with Macros: A Programming Aid for the HEP and other Parallel Processors," Argonne National Laboratory, Mathematics and Computer Science Division, Technical Report No. ANL-83-97, Dec. 1983.

APPENDIX B

[original]

Original (CDC?) Sequential GAMTEB

```
      C HERE COMES SGAMTEB (SEQUENTIAL VERSION ..... THE ORIGINAL)
      C
            PROGRAM GAMTEB(OUTPUT,TAPE4=OUTPUT)
      C
      C           SCALAR MONTE CARLO CODE TO TRANSPORT .001 TO 20.0 MEV
      C           GAMMA RAYS IN A CARBON CYLINDER OF LENGTH CL, RADIUS CRAD
      C
            COMMON X,Y,Z,U,V,W,ERG,IA,WT,NP,UOLD,VOLD,WOLD,CL,CL2,CRAD2,DLS,JA
            DIMENSION E(35),EL(35),XC(35),XPP(35),XPE(35),TRANS(35),
           1   BSCAT(35),ESCAPE(35),TRANS2(35),BSCAT2(35),ESCAPE2(35)
           2 ,RTRANS(35),RBSCAT(35),RESCAPE(35)
            DIMENSION BANK(100,10),PBL(10),FIM(2),IBANK(100,10)
            DIMENSION TRANSI(35),BSCATI(35),ESCAPEI(35)
            EQUIVALENCE(PBL,X),(BANK,IBANK)
            INTEGER CUTOFF
            DATA RHO/2.22/
            DATA (E(I),I=1,35)/ .001, .0015, .002, .003, .004, .005,
           1      .006, .008, .01, .015, .02, .03, .04, .05, .06, .08,
           2      .1, .15, .2, .3, .4, .5, .6, .8, 1., 1.5, 2., 3., 4., 5.,
           3      6., 8., 10., 15., 20./
            DATA (XC(I),I=1,35)/.0150,.0296,.0451,.0717,.0913
           1,.105,.115,.128,.137,.152
           2,.160,.165,.165,.163,.160
           3,.153,.146,.133,.122,.106,.0953,.0867,.0802,.0707,.0637
           4,.0516,.044,.0346,.0289,.0250,.0221,.0181,.0154,.0114,.00913/
            DATA (XPP(I),I=1,35)/ 0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,
           1      0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,.0000792,
           2      .000316, .000923, .00153, .00208, .00256, .00343,
           3      .00414, .00547, .00652/
            DATA (XPE(I),I=1,35)/ 2010., 632., 280., 87.7, 37.3, 18.9,
           1      10.4, 4.01, 1.91, .489, .192, .0491, .0186, .00887,
           2      .00481, .00179, .000862, .000234, .0000918,
           4  0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0./
      C
      C           CONVERT CROSS-SECTION UNITS TO BE PER CM.
            DO 1 I=1,35
            XC(I)=ALOG( XC(I)*RHO )
            IF(XPP(I).EQ.0.) XPP(I)=1.0E-123
            IF(XPE(I).EQ.0.) XPE(I)=1.0E-123
            XPP(I)=ALOG( XPP(I)*RHO )
            XPE(I)=ALOG( XPE(I)*RHO )
            EL(I)=ALOG(E(I))
          1 CONTINUE
      C
      C           INITIALIZE PROBLEM INPUT
            NPP=500000
            NCOL=0
            WCP1=.5
            WCP2=.25
            NCO=0
            WCO=0
            WCP=0
            EC = .001
            NPS = 0
            KRN = 123454321
            CL=20.0
```

```
      CL2=CL+10
      CRAD=1.0
      CRAD2=CRAD**2
      CUTOFF = 0
      BTOT=0.0
      BTOT2=0.0
      TTOT=0.0
      TTOT2=0.0
      ETOT=0.0
      ETOT2=0.0
      WRL=0.
      WRG=0.
      NR=0
      INBNK=0
      NBANK=0
      FIM(1)=1
      FIM(2)=2.0
      INBNK=0
      DO 5 I = 1,35
      TRANS2(I)= 0.0
      BSCAT2(I)= 0.0
      ESCAPE2(I)= 0.0
      TRANS(I) = 0.0
      BSCAT(I) = 0.0
    5 ESCAPE(I) = 0.0
C
C        START A HISTORY
      CALL SECOND(TO)
   10 NPS = NPS + 1
      IF(NPS.EQ.NPP+1) GO TO 140
C
C        SET SOURCE VALUES
      ERG = 6.0
      WT = 1.0
      U = 0.0
      V = 1.0
      W = 0.0
      X = 0.0
      Y = .000001
      Z = 0.0
      JA = 0
      IA=1
C
C        CALCULATE DISTANCE DLS TO NEXT SURFACE INTERSECTION
C        FOR ALL THREE SURFACES AND ALSO THE NUMBER JA OF THE
C        NEXT SURFACE INTERSECTED
   20 JA=0
      CALL TRACK
C
C        FIND ENERGY POINTER FOR CROSS SECTIONS AND TALLYS
      DO 30 IE = 1,35
      IF(ERG.GT.E(IE)) GO TO 30
      I = IE
      GO TO 31
   30 CONTINUE
C
```

```
113  C          INTERPOLATION TO GET CROSS SECTIONS AS F(ERG)
114     31 F=(ALOG(ERG)-EL(I-1))/(EL(I)-EL(I-1))
115        XSC=EXP( XC(I-1)+F*(XC(I)-XC(I-1)) )
116        XSPP=EXP( XPP(I-1)+F*(XPP(I)-XPP(I-1)) )
117        XSPE=EXP( XPE(I-1)+F*(XPE(I)-XPE(I-1)) )
118        XST  = XSC + XSPP + XSPE
119  C
120  C          CALCULATE DISTANCE TO NEXT COLLISION
121        S = -ALOG(RANF(KRN))/XST
122  C
123  C          SEE IF COLLISION IS STILL INSIDE CYLINDER
124  C          IF NOT, DO TALLYS; IF SO, DO COLLISION PHYSICS
125        IF(S.LT.DLS) GO TO 60
126        X=X+U*DLS
127        Y=Y+V*DLS
128        Z=Z+W*DLS
129        GO TO(42,50,53,52) JA
130     42 BSCATI(I) = BSCATI(I) + WT
131        BTOTI = BTOTI + WT
132        GO TO 11
133     52 TRANSI(I) = TRANSI(I) + WT
134        TTOTI = TTOTI + WT
135        GO TO 11
136     50 ESCAPEI(I) = ESCAPEI(I) + WT
137        ETOTI = ETOTI + WT
138        GO TO 11
139  C  CROSS INTERNAL SURFACE  SPLIT OR ROULETTE
140     53 IAP=IA
141        IA=2-IA/2
142        T1=FIM(IA)/FIM(IAP)
143        IF(T1.GT.1.0) GO TO 57
144  C  RUSSIAN ROULETTE
145        IF(T1.LT.RANF(KRN)) GO TO 58
146        WTSAV=WT
147        WT=WT/T1
148        WRG=WRG+(WT-WTSAV)
149        GO TO 20
150  C  KILLED IN RUSSIAN ROULETTE
151     58 WRL=WRL+WT
152        NR=NR+1
153        GO TO 11
154  C  SPLITTING
155     57 NP=T1-1.
156        WT=WT/T1
157        NS=NS+NP
158        NBANK=NBANK+NP
159        INBNK=INBNK+1
160        DO 59 IX=1,10
161     59 BANK(INBNK,IX)=PBL(IX)
162        GO TO 20
163  C  CHECK BANK BEFORE STARTING NEW PARTICLE
164     11 IF(NBANK.EQ.0) GO TO 234
165        DO 521 IX=1,9
166    521 PBL(IX)=BANK(INBNK,IX)
167        NBANK=NBANK-1
168        IBANK(INBNK,10)=IBANK(INBNK,10)-1
```

```
      IF(IBANK(INBNK,10).EQ.0) INBNK=INBNK-1
      GO TO 20
C
C        COLLISIONS
   60 JA = 0
      X=X+U*S
      Y=Y+V*S
      Z=Z+W*S
      NCOL=NCOL+1
C  SURVIVAL BIAS
      WTSAV=WT
      WT=WT*(1.-XSPE/XST)
      ABSORB=ABSORB+(WTSAV-WT)
      XSTSB=XST-XSPE
C WEIGHT CUTOFF
      IF(WT.GT.WCP2) GO TO 832
      IF(WT*FIM(IA).LT.RANF(KRN)*WCP1*FIM(1)) GO TO 642
      WTSAV=WT
      WT=WCP1*FIM(1)/FIM(IA)
      WCP=WCP+(WT-WTSAV)
  832 CONTINUE
      IF(RANF(KRN).GE.XSC/XSTSB) GO TO 100
      T1 = 1.956917*ERG
C        GET NEW ENERGY T4 AND COMPTON SCATTERING ANGLE
      CALL KLEIN(T1,T4)
      CSA = 1.+1./T1-1./T4
      T5 = .511008*T4
      IF(ABS(CSA).GT.1.) CSA=SIGN(1.,CSA)
      ERG = T5
C
C        SEE IF NEW ENERGY IS LESS THAN CUTOFF
      IF(ERG.GT.EC) GO TO 70
      CUTOFF = CUTOFF + 1
      GO TO 11
C        MAKE COMPTON ANGLE RELATIVE TO PROBLEM COORDINATE SYSTEM
   70 UOLD = U
      VOLD = V
      WOLD = W
      CALL ROTAS(CSA)
      GO TO 20
C
C        PAIR PRODUCTION
  100 ERG = 0.511008
      WT = 2.*WT
C
C        CHECK ENERGY CUTOFF
      IF(ERG.GT.EC) GO TO 110
      CUTOFF = CUTOFF + 1
      GO TO 11
C
C        ISOTROPIC EMISSION IN LAB SYSTEM
  110 CALL ISOS
      GO TO 20
C
C        PHOTOELECTRIC ABSORPTION
C NOW HANDLED BY SURVIVAL BIASING
```

```
C 130 ABSORB = ABSORB + WT
C       GO TO 11
C  TERMINATE PARTICLE TO WEIGHT CUTOFF
  642 WCO=WCO+WT
      NCO=NCO+1
      GO TO 11
  234 DO 829 I=1,35
      BSCAT(I)=BSCAT(I)+BSCATI(I)
      BSCAT2(I)=BSCAT2(I)+BSCATI(I)**2
      TRANS(I)=TRANS(I)+TRANSI(I)
      TRANS2(I)=TRANS2(I)+TRANSI(I)**2
      ESCAPE(I)=ESCAPE(I)+ESCAPEI(I)
      ESCAPE2(I)=ESCAPE2(I)+ESCAPEI(I)**2
      BSCATI(I)=0.
      TRANSI(I)=0.
      ESCAPEI(I)=0.
  829 CONTINUE
      BTOT=BTOT+BTOTI
      TTOT=TTOT+TTOTI
      ETOT=ETOT+ETOTI
      BTOT2=BTOT2+BTOTI**2
      TTOT2=TTOT2+TTOTI**2
      ETOT2=ETOT2+ETOTI**2
      BTOTI=0.
      TTOTI=0.
      ETOTI=0.
      GO TO 10
C
C         PRINT OUTPUT
  140 NPS = NPS - 1
      CALL SECOND(T1)
      TEND = T1 - TO
      WRITE(4,7634) NCOL
 7634 FORMAT(5H NCOL,I10)
      WRITE(4,1401)
 1401 FORMAT(7HSCALERT,/)
      WRITE(4,150) NPS
  150 FORMAT(6HNPS = ,I6)
      WRITE(4,200)
  200 FORMAT(///,8X,1HE,13X,5HBSCAT,9X,9HREL ERROR)
      DO 220 I=1,35
      RNPS= NPS
      TRANS(I) = TRANS(I)/RNPS
      BSCAT(I) = BSCAT(I)/RNPS
      ESCAPE(I) = ESCAPE(I)/RNPS
      TRANS2(I)=TRANS2(I)/RNPS
      BSCAT2(I)=BSCAT2(I)/RNPS
      ESCAPE2(I)=ESCAPE2(I)/RNPS
      IF(TRANS(I).NE.0.0)GO TO 203
      RTRANS(I)= 0.0
      GO TO 204
  203 RTRANS(I)= SQRT((TRANS2(I)-TRANS(I)**2)/ RNPS)
      RTRANS(I)= RTRANS(I)/TRANS(I)
  204 IF(BSCAT(I).NE.0.0)GO TO 205
      RBSCAT(I)= 0.0
      GO TO 206
```

```
      205 RBSCAT(I)= SQRT((BSCAT2(I)-BSCAT(I)**2)/ RNPS)
          RBSCAT(I)= RBSCAT(I)/ BSCAT(I)
      206 IF(ESCAPE(I). NE. 0. 0)GO TO 207
          RESCAPE(I)= 0. 0
          GO TO 209
      207 RESCAPE(I)= SQRT((ESCAPE2(I)-ESCAPE(I)**2)/ RNPS)
          RESCAPE(I)= RESCAPE(I)/ ESCAPE(I)
      209 WRITE(4,210) E(I),BSCAT(I),RBSCAT(I)
      220 CONTINUE
          TTOT = TTOT/RNPS
          TTOT2 = TTOT2/RNPS
          BTOT = BTOT/RNPS
          BTOT2 = BTOT2/RNPS
          ETOT = ETOT/RNPS
          ETOT2 = ETOT2/RNPS
          IF(TTOT. NE. 0. 0) GO TO 2000
          RTTOT = 0. 0
          GO TO 2001
     2000 RTTOT = SQRT((TTOT2 - TTOT**2)/RNPS)
          RTTOT = RTTOT/TTOT
     2001 IF(BTOT. NE. 0. 0) GO TO 2002
          RBTOT = 0. 0
          GO TO 2003
     2002 RBTOT = SQRT((BTOT2 - BTOT**2)/RNPS)
          RBTOT = RBTOT/BTOT
     2003 IF(ETOT. NE. 0. 0) GO TO 2004
          RETOT = 0. 0
          GO TO 2005
     2004 RETOT = SQRT((ETOT2 - ETOT**2)/RNPS)
          RETOT = RETOT/ETOT
     2005 CONTINUE
          WRITE(4,2020) BTOT, RBTOT
     2020 FORMAT(/,6X,5HTOTAL,9X,1PE10. 3,5X,OPF7. 4)
          WRITE(4,201)
      201 FORMAT(///,8X,1HE,13X,6HESCAPE,8X,9HREL ERROR)
          DO 225 I=1,35
          WRITE(4,210) E(I),ESCAPE(I),RESCAPE(I)
      225 CONTINUE
          WRITE(4,2020) ETOT, RETOT
          WRITE(4,202)
      202 FORMAT(///,8X,1HE,13X,5HTRANS,9X,9HREL ERROR)
          DO 230 I=1,35
          WRITE(4,210) E(I),TRANS(I),RTRANS(I)
      210 FORMAT(5X,1PE10. 3,5X,1PE10. 3,5X,OPF6. 3)
      230 CONTINUE
          WRITE(4,2020) TTOT, RTTOT
          ABSORB = ABSORB/NPS
          WRG=WRG/NPP
          WRL=WRL/NPP
          WCP=WCP/NPP
          WCO=WCO/NPP
          WRITE(4,221) ABSORB, CUTOFF
      221 FORMAT(////,9HABSORB = ,1PE10. 3,5X,9HCUTOFF = ,I5)
          WRITE(4,3728) NS,NR
     3728 FORMAT(28H TRACKS CREATED BY SPLITTING,I8,
         1 24H TRACKS LOST TO ROULETTE,I8)
```

```
      WRITE(4,3729) WRG,WRL
 3729 FORMAT(27H WEIGHT CREATED BY ROULETTE,1PE11.4,
     1 24H WEIGHT LOST TO ROULETTE,1PE11.4)
      WRITE(4,3730) NCO
 3730 FORMAT(29H TRACKS LOST TO WEIGHT CUTOFF,I8)
      WRITE(4,3731) WCP,WCO
 3731 FORMAT(32H WEIGHT CREATED BY WEIGHT CUTOFF,1PE11.4,
     1 29H WEIGHT LOST TO WEIGHT CUTOFF,1PE11.4)
      WRITE(4,2021) TEND
 2021 FORMAT(///, 13HTOTAL TIME = ,1PE10.3, 8H SECONDS)
      STOP
      END

      SUBROUTINE TRACK
C        CALCULATE ALL INTERSECTIONS WITH ALL THREE SURFACES
      COMMON X,Y,Z,U,V,W,ERG,IA,WT,NP,UOLD,VOLD,WOLD,CL,CL2,CRAD2,DLS,JA
      DLSS = 1.0E10
      IF(IA.EQ.2) GO TO 19
      DO 300 J=1,3
      D1 = -1.0
      GO TO (55,160,50),J
   50 IF(V.EQ.0.) GO TO 300
      D1 = (CL-Y)/V
      GO TO 280
   55 IF(V.EQ.0.) GO TO 300
      D1 = -Y/V
      GO TO 280
  160 T1 = U**2 + W**2
      IF(T1.EQ.0.) GO TO 300
      A1 = (X*U + Z*W)/T1
      B1 = (X**2 + Z**2 - CRAD2)/T1
      T1 = A1**2 - B1
      IF(T1.LT.0.) GO TO 300
      T2 = SQRT(T1)
      D1 = -A1 + T2
      D2 = -A1 - T2
      IF(J.EQ.JA) D2=D1=-2.*A1
      GO TO 290
  280 D2 = -D1
  290 IF(D1.LE.0.) GO TO 300
      IF(D2.GT.0.) D1=D2
      IF(D1.GE.DLSS) GO TO 300
      JAS = J
      DLSS = D1
  300 CONTINUE
      DLS = DLSS+1.0E-10
      JA = JAS
      RETURN
   19 DO 301 J=2,4
      D1 = -1.0
      GO TO (56,161,51,56),J
   51 IF(V.EQ.0.) GO TO 301
      D1 = (CL-Y)/V
      GO TO 281
   56 IF(V.EQ.0.) GO TO 301
      D1 = (CL2-Y)/V
```

```
      GO TO 281
  161 T1 = U**2 + W**2
      IF(T1.EQ.0.) GO TO 301
      A1 = (X*U + Z*W)/T1
      B1 = (X**2 + Z**2 - CRAD2)/T1
      T1 = A1**2 - B1
      IF(T1.LT.0.) GO TO 301
      T2 = SQRT(T1)
      D1 = -A1 + T2
      D2 = -A1 - T2
      IF(J.EQ.JA) D2=D1=-2.*A1
      GO TO 291
  281 D2 = -D1
  291 IF(D1.LE.0.) GO TO 301
      IF(D2.GT.0.) D1=D2
      IF(D1.GE.DLSS) GO TO 301
      JAS = J
      DLSS = D1
  301 CONTINUE
      DLS = DLSS+1.0E-10
      JA = JAS
      RETURN
      END

      SUBROUTINE KLEIN(T1,T4)
C        SAMPLE FROM KLEIN-NISHINA USING INVERSE FIT.
C        T1=ENERGY IN, T4=ENERGY OUT, IN UNITS OF THE REST MASS
C        OF AN ELECTRON.
C
      RN=RANF(KRN)
      T2=1./T1
      T4=2.*T1+1.
      T5=1./T4
      T6=ALOG(T4)
      T3=2.*T1*(1.+T1)*T5**2+4.*T2+(1.-2.*T2*(1.+T2))*T6
      IF(T1.LE.1.16666667)GO TO 20
      T7=1.65898+T2*(.62537*T2-1.00796)
      T3=T7/T3
      IF(RN.LE.T3)GO TO 10
      T4=(T6-1.20397)/(1.-T3)
      T7=.3*EXP(T4*(T3-RN))
      GO TO 40
   10 T4=T7/(3.63333+T2*(5.44444*T2-4.66667))
      T7=.5*T7
      T2=RN/T3
      T3=2.1
      T5=1.4
      GO TO 30
   20 T4=T3/(T4+T5)
      T7=.5*T3
      T2=RN
      T5=1.-T5
      T3=3.*T5
      T5=2.*T5
   30 T7=1.+T2*(T2*(2.*T7+T4-T3+T2*(T5-T7-T4))-T7)
   40 T4=T7*T1
```

```
      RETURN
      END

      SUBROUTINE ISOS
C        SAMPLE A DIRECTION U,V,W ISOTROPICALLY.
C
      COMMON X,Y,Z,U,V,W,ERG,IA,WT,NP,UOLD,VOLD,WOLD,CL,CL2,CRAD2,DLS,JA
   10 T1=2.*RANF(KRN)-1.
      T2=2.*RANF(KRN)-1.
      RSQ=T1**2+T2**2
      IF(RSQ.GT.1.0)GO TO 10
      U=2.*RSQ-1.
      T3=SQRT((1.-U**2)/RSQ)
      V=T1*T3
      W=T2*T3
      RETURN
      END

      SUBROUTINE ROTAS(C)
C        ROTATE UOLD,VOLD,WOLD TO U,V,W THROUGH A POLAR
C        ANGLE WHOSE COSINE IS C, AND THROUGH AN AZIMUTHAL
C        ANGLE SAMPLED UNIFORMLY.
C
      COMMON X,Y,Z,U,V,W,ERG,IA,WT,NP,UOLD,VOLD,WOLD,CL,CL2,CRAD2,DLS,JA
   10 T1=2.*RANF(KRN)-1.
      T2=2.*RANF(KRN)-1.
      R=T1**2+T2**2
      IF(R.GT.1.0)GO TO 10
      R=SQRT((1.-C**2)/R)
      T1=T1*R
      T2=T2*R
      IF(ABS(WOLD).GT..999999)GO TO 30
      S=SQRT(1.-WOLD**2)
      U=UOLD*C+(T1*UOLD*WOLD-T2*VOLD)/S
      V=VOLD*C+(T1*VOLD*WOLD+T2*UOLD)/S
      W=WOLD*C-T1*S
      RETURN
   30 U=T1
      V=T2
      W=WOLD*C
      RETURN
      END
```

APPENDIX C

[f77]

FORTRAN77 Sequential GAMTEB

KEY TO ANNOTATIONS:

PROGRAMS

p00: Scalar Monte Carlo Transport Code
p10: set up problem constants
p12: initialize run parameters
p13: set up for particle counting
p14: format and print report
p16: generate random seed for next particle
p17: run history for one particle and offspring
p18: check for run completion

DATA PATHS

d03: problem constants
d04: converted cross-section tables
d06: run statistics
d13: first random number seed

CHANGES

OLD: original code
NEW: miscellaneous changes needed for the VAX
ALG CH: Monte Carlo random number generator installed
BUGFIX: REAL/INTEGER equivalence replaced by INTEGER/INTEGER and REAL/REAL equivalences, KRN put in COMMON
RENAME: variable names reduced to 6 characters
EXTRA: removed extra initialization
STYLE: REAL initialized with REAL value

```
      C HERE COMES SGAMTEB (SEQUENTIAL VERSION     THE ORIGINAL)
      C
      C     PROGRAM GAMTEB(OUTPUT,TAPE4=OUTPUT)                          p00 OLD
            PROGRAM GAMTEB                                                 NEW
      C
      C        SCALAR MONTE CARLO CODE TO TRANSPORT .001 TO 20.0 MEV
      C        GAMMA RAYS IN A CARBON CYLINDER OF LENGTH CL, RADIUS CRAD  p00
      C
            COMMON X,Y,Z,U,V,W,ERG,WT,IA,NP                                BUGFIX
                                   d05
            COMMON UOLD,VOLD,WOLD,CL,CL2,CRAD2,DLS,JA                      BUGFIX
                                        d03
            COMMON /NEW/ KRN                                           d13 BUGFIX
            DIMENSION FIM(2),E(35)                                     d03
            DIMENSION EL(35),XC(35),XPP(35),XPE(35)                    d04
            DIMENSION TRANS(35),BSCAT(35),ESCAPE(35),
           1   TRANS2(35),BSCAT2(35),ESCAP2(35),                       d06 RENAME
           2   TRANSI(35),BSCATI(35),ESCAPI(35)                        d06 RENAME
            DIMENSION RTRANS(35),RBSCAT(35),RESCAP(35)                 p14 RENAME
            DIMENSION BANK(100,8),PBL(8),IBANK(100,2),IPBL(2)          p17 BUGFIX
            EQUIVALENCE(PBL,X),(IPBL,IA)                                   BUGFIX
            INTEGER CUTOFF                                             p17
            DATA RHO/2.22/                                             d03
            DATA (E(I),I=1,35)/ .001, .0015, .002, .003, .004, .005,
           1     .006, .008, .01, .015, .02, .03, .04, .05, .06, .08,
           2     .1, .15, .2, .3, .4, .5, .6, .8, 1., 1.5, 2., 3., 4., 5.,
           3     6., 8., 10., 15., 20./                                d03
            DATA (XC(I),I=1,35)/.0150,.0296,.0451,.0717,.0913          d04
           1,.105,.115,.128,.137,.152
           2,.160,.165,.165,.163,.160
           3,.153,.146,.133,.122,.106,.0953,.0867,.0802,.0707,.0637
           4,.0516,.044,.0346,.0289,.0250,.0221,.0181,.0154,.0114,.00913/
            DATA (XPP(I),I=1,35)/ 0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,
           1     0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,.0000792,
           2     .000316, .000923, .00153, .00208, .00256, .00343,
           3     .00414, .00547, .00652/
            DATA (XPE(I),I=1,35)/ 2010., 632., 280., 87.7, 37.3, 18.9,
           1     10.4, 4.01, 1.91, .489, .192, .0491, .0186, .00887,
           2     .00481, .00179, .000862, .000234, .0000918,
           4  0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0./           d04
      C
      C        CONVERT CROSS-SECTION UNITS TO BE PER CM.              p10
            DO 1 I=1,35
            XC(I)=ALOG( XC(I)*RHO )
      C     IF(XPP(I).EQ.0.) XPP(I)=1.0E-123                           p10 OLD
```

```
                IF(XPP(I).EQ.0.) XPP(I)=1.0E-37                          p10 NEW
C               IF(XPE(I).EQ.0.) XPE(I)=1.0E-123                          |  OLD
                IF(XPE(I).EQ.0.) XPE(I)=1.0E-37                           |  NEW
                XPP(I)=ALOG( XPP(I)*RHO )                                 |
                XPE(I)=ALOG( XPE(I)*RHO )                                 |
                EL(I)=ALOG(E(I))                                          |
              1 CONTINUE                                                  p10
C
C                  INITIALIZE PROBLEM INPUT                               p12
C               NPP=500000                                                |  OLD
                NPP=100                                                   |  NEW
                NCOL=0                                                    p12

                WCP1=.5                                                   p10
                WCP2=.25                                                  p10

                NCD=0                                                     p12
                WCD=0                                                     |
                WCP=0                                                     p12

                EC = .001                                                 p10

                NPS = 0                                                   p13
C               KRN = 123454321                                           |  OLD
                KRN2 = 123                                                p13 ALG CH

                CL=20.0                                                   p10
                CL2=CL+10.                                                |
                CRAD=1.0                                                  |
                CRAD2=CRAD**2                                             p10

                CUTOFF = 0                                                p12
                BTOT=0.0                                                  |
                BTOT2=0.0                                                 |
                TTOT=0.0                                                  |
                TTOT2=0.0                                                 |
                ETOT=0.0                                                  |
                ETOT2=0.0                                                 |
                WRL=0.                                                    |
                WRG=0.                                                    |
                NR=0                                                      p12

                INBNK=0                                                   p17
                NBANK=0                                                   p17

C               FIM(1)=1                                                  p10 OLD
                FIM(1)=1.0                                                |  STYLE
                FIM(2)=2.0                                                p10

C               INBNK=0                                                   p17 EXTRA

                DO 5 I = 1,35                                             p12
                TRANS2(I)= 0.0                                            |
                BSCAT2(I)= 0.0                                            |
                ESCAP2(I)= 0.0                                            |  RENAME
                TRANS(I) = 0.0                                            p12
```

```
113          BSCAT(I) = 0.0                                              p12
114        5 ESCAPE(I) = 0.0                                             p12
115  C
116  C          START A HISTORY                                         p16
117          CALL SECOND(TO)
118       10 NPS = NPS + 1                                          p16 & p18
119          IF(NPS.EQ.NPP+1) GO TO 140                             p16 & p18
120  C
121  C          SET SOURCE VALUES                                       p10
122          ERG = 6.0                                                  !
123          WT = 1.0                                                   !
124          U = 0.0                                                    !
125          V = 1.0                                                    !
126          W = 0.0                                                    !
127          X = 0.0                                                    !
128          Y = .000001                                                !
129          Z = 0.0                                                 p10
130          JA = 0
131          IA=1                                                    p10
132
133  C       GENERATE NEW RANDOM SEED                                p16 ALG CH
134          XJUNK = RANDO(KRN2)                                        !  ALG CH
135          KRN = KRN2                                              p16 ALG CH
136  C
137  C          CALCULATE DISTANCE DLS TO NEXT SURFACE INTERSECTION   p17
138  C          FOR ALL THREE SURFACES AND ALSO THE NUMBER JA OF THE       !
139  C          NEXT SURFACE INTERSECTED                                   !
140       20 JA=0                                                          !
141          CALL TRACK                                                    !
142  C                                                                     !
143  C          FIND ENERGY POINTER FOR CROSS SECTIONS AND TALLYS          !
144          DO 30 IE = 1,35                                               !
145          IF(ERG.GT.E(IE)) GO TO 30                                     !
146          I = IE                                                        !
147          GO TO 31                                                      !
148       30 CONTINUE                                                      !
149  C                                                                     !
150  C          INTERPOLATION TO GET CROSS SECTIONS AS F(ERG)               !
151       31 F=(ALOG(ERG)-EL(I-1))/(EL(I)-EL(I-1))                         !
152          XSC=EXP( XC(I-1)+F*(XC(I)-XC(I-1)) )                          !
153          XSPP=EXP( XPP(I-1)+F*(XPP(I)-XPP(I-1)) )                      !
154          XSPE=EXP( XPE(I-1)+F*(XPE(I)-XPE(I-1)) )                      !
155          XST = XSC + XSPP + XSPE                                       !
156  C                                                                     !
157  C          CALCULATE DISTANCE TO NEXT COLLISION                       !
158          S = -ALOG(RANF(KRN))/XST                                      !
159  C                                                                     !
160  C          SEE IF COLLISION IS STILL INSIDE CYLINDER                  !
161  C          IF NOT, DO TALLYS; IF SO, DO COLLISION PHYSICS             !
162          IF(S.LT.DLS) GO TO 60                                         !
163          X=X+U*DLS                                                     !
164          Y=Y+V*DLS                                                     !
165          Z=Z+W*DLS                                                     !
166          GO TO(42,50,53,52) JA                                         !
167       42 BSCATI(I) = BSCATI(I) + WT                                    !
168          BTOTI = BTOTI + WT                                         p17
```

```
                GO TO 11                                                    p17
     52 TRANSI(I) = TRANSI(I) + WT
        TTOTI = TTOTI + WT
        GO TO 11
     50 ESCAPI(I) = ESCAPI(I) + WT                                          RENAME
        ETOTI = ETOTI + WT
        GO TO 11
C  CROSS INTERNAL SURFACE  SPLIT OR ROULETTE
     53 IAP=IA
        IA=2-IA/2
        T1=FIM(IA)/FIM(IAP)
        IF(T1.GT.1.0) GO TO 57
C  RUSSIAN ROULETTE
        IF(T1.LT.RANF(KRN)) GO TO 58
        WTSAV=WT
        WT=WT/T1
        WRG=WRG+(WT-WTSAV)
        GO TO 20
C  KILLED IN RUSSIAN ROULETTE
     58 WRL=WRL+WT
        NR=NR+1
        GO TO 11
C  SPLITTING
     57 NP=T1-1.
        WT=WT/T1
        NS=NS+NP
        NBANK=NBANK+NP
        INBNK=INBNK+1
        DO 59 IX=1,8                                                        BUGFIX
     59 BANK(INBNK,IX)=PBL(IX)                                              BUGFIX
        DO 61 IX=1,2                                                        BUGFIX
     61 IBANK(INBNK,IX)=IPBL(IX)                                            BUGFIX
        GO TO 20
C  CHECK BANK BEFORE STARTING NEW PARTICLE
     11 IF(NBANK.EQ.0) GO TO 234
        DO 521 IX=1,8                                                       BUGFIX
    521 PBL(IX)=BANK(INBNK,IX)                                              BUGFIX
        DO 522 IX=1,1                                                       BUGFIX
    522 IPBL(IX)=IBANK(INBNK,IX)                                            BUGFIX
        NBANK=NBANK-1
        IBANK(INBNK,2)=IBANK(INBNK,2)-1                                     BUGFIX
        IF(IBANK(INBNK,2).EQ.0) INBNK=INBNK-1                               BUGFIX
        GO TO 20
C
C          COLLISIONS
     60 JA = 0
        X=X+U*S
        Y=Y+V*S
        Z=Z+W*S
        NCOL=NCOL+1
C  SURVIVAL BIAS
        WTSAV=WT
        WT=WT*(1.-XSPE/XST)
        ABSORB=ABSORB+(WTSAV-WT)
        XSTS3=XST-XSPE
C WEIGHT CUTOFF                                                             p17
```

```
      IF(WT.GT.WCP2) GO TO 832                                          p17
      IF(WT*FIM(IA).LT.RANF(KRN)*WCP1*FIM(1)) GO TO 642
      WTSAV=WT
      WT=WCP1*FIM(1)/FIM(IA)
      WCP=WCP+(WT-WTSAV)
  832 CONTINUE
      IF(RANF(KRN).GE.XSC/XSTSB) GO TO 100
      T1 = 1.956917*ERG
C        GET NEW ENERGY T4 AND COMPTON SCATTERING ANGLE
      CALL KLEIN(T1,T4)
      CSA = 1.+1./T1-1./T4
      T5 = .511008*T4
      IF(ABS(CSA).GT.1.) CSA=SIGN(1.,CSA)
      ERG = T5
C
C        SEE IF NEW ENERGY IS LESS THAN CUTOFF
      IF(ERG.GT.EC) GO TO 70
      CUTOFF = CUTOFF + 1
      GO TO 11
C        MAKE COMPTON ANGLE RELATIVE TO PROBLEM COORDINATE SYSTEM
   70 UOLD = U
      VOLD = V
      WOLD = W
      CALL ROTAS(CSA)
      GO TO 20
C
C        PAIR PRODUCTION
  100 ERG = 0.511008
      WT = 2.*WT
C
C        CHECK ENERGY CUTOFF
      IF(ERG.GT.EC) GO TO 110
      CUTOFF = CUTOFF + 1
      GO TO 11
C
C        ISOTROPIC EMISSION IN LAB SYSTEM
  110 CALL ISOS
      GO TO 20
C
C        PHOTOELECTRIC ABSORPTION
C NOW HANDLED BY SURVIVAL BIASING
C 130 ABSORB = ABSORB + WT
C     GO TO 11
C  TERMINATE PARTICLE TO WEIGHT CUTOFF
  642 WCO=WCO+WT
      NCO=NCO+1
      GO TO 11
  234 DO 829 I=1,35
      BSCAT(I)=BSCAT(I)+BSCATI(I)
      BSCAT2(I)=BSCAT2(I)+BSCATI(I)**2
      TRANS(I)=TRANS(I)+TRANSI(I)
      TRANS2(I)=TRANS2(I)+TRANSI(I)**2
      ESCAPE(I)=ESCAPE(I)+ESCAPI(I)                                     RENAME
      ESCAP2(I)=ESCAP2(I)+ESCAPI(I)**2                                  RENAME
      BSCATI(I)=0.
      TRANSI(I)=0.                                                      p17
```

```
281          ESCAPI(I)=0                                                    p17 RENAME
282       829 CONTINUE
283          BTOT=BTOT+BTOTI
284          TTOT=TTOT+TTOTI
285          ETOT=ETOT+ETOTI
286          BTOT2=BTOT2+BTOTI**2
287          TTOT2=TTOT2+TTOTI**2
288          ETOT2=ETOT2+ETOTI**2
289          BTOTI=0.
290          TTOTI=0.
291          ETOTI=0.
292          GO TO 10                                                       p17
293     C
294     C          PRINT OUTPUT                                             p14
295       140 NPS = NPS - 1                                                  p14
296          CALL SECOND(T1)
297          TEND = T1 - T0
298          WRITE(4,7634) NCOL                                             p14
299     7634 FORMAT(5H NCOL,I10)
300          WRITE(4,1401)
301       1401 FORMAT(7HSCALERT,/)
302          WRITE(4,150) NPS
303        150 FORMAT(6HNPS = ,I6)
304          WRITE(4,200)
305        200 FORMAT(///,8X,1HE,13X,5HBSCAT,9X,9HREL ERROR)
306          DO 220 I=1,35
307          RNPS= NPS
308          TRANS(I) = TRANS(I)/RNPS
309          BSCAT(I) = BSCAT(I)/RNPS
310          ESCAPE(I) = ESCAPE(I)/RNPS
311          TRANS2(I)=TRANS2(I)/RNPS
312          BSCAT2(I)=BSCAT2(I)/RNPS
313          ESCAP2(I)=ESCAP2(I)/RNPS                                       RENAME
314          IF(TRANS(I).NE.0.0)GO TO 203
315          RTRANS(I)= 0.0
316          GO TO 204
317       203 RTRANS(I)= SQRT((TRANS2(I)-TRANS(I)**2)/.RNPS)
318          RTRANS(I)= RTRANS(I)/TRANS(I)
319       204 IF(BSCAT(I).NE.0.0)GO TO 205
320          RBSCAT(I)= 0.0
321          GO TO 206
322       205 RBSCAT(I)= SQRT((BSCAT2(I)-BSCAT(I)**2)/ RNPS)
323          RBSCAT(I)= RBSCAT(I)/ BSCAT(I)
324       206 IF(ESCAPE(I).NE.0.0)GO TO 207
325          RESCAP(I)= 0.0                                                 RENAME
326          GO TO 209
327       207 RESCAP(I)= SQRT((ESCAP2(I)-ESCAPE(I)**2)/ RNPS)               RENAME
328          RESCAP(I)= RESCAP(I)/ ESCAPE(I)                                RENAME
329       209 WRITE(4,210) E(I),BSCAT(I),RBSCAT(I)
330       220 CONTINUE
331          TTOT = TTOT/RNPS
332          TTOT2 = TTOT2/RNPS
333          BTOT = BTOT/RNPS
334          BTOT2 = BTOT2/RNPS
335          ETOT = ETOT/RNPS
336          ETOT2 = ETOT2/RNPS                                             p14
```

```
      IF(TTOT.NE.0.0) GO TO 2000                                        p14
      RTTOT = 0.0
      GO TO 2001
 2000 RTTOT = SQRT((TTOT2 - TTOT**2)/RNPS)
      RTTOT = RTTOT/TTOT
 2001 IF(BTOT.NE.0.0) GO TO 2002
      RBTOT = 0.0
      GO TO 2003
 2002 RBTOT = SQRT((BTOT2 - BTOT**2)/RNPS)
      RBTOT = RBTOT/BTOT
 2003 IF(ETOT.NE.0.0) GO TO 2004
      RETOT = 0.0
      GO TO 2005
 2004 RETOT = SQRT((ETOT2 - ETOT**2)/RNPS)
      RETOT = RETOT/ETOT
 2005 CONTINUE
      WRITE(4,2020) BTOT, RBTOT
 2020 FORMAT(/,6X,5HTOTAL,9X,1PE10.3,5X,0PF7.4)
      WRITE(4,201)
  201 FORMAT(///,8X,1HE,13X,6HESCAPE,8X,9HREL ERROR)
      DO 225 I=1,35
      WRITE(4,210) E(I),ESCAPE(I),RESCAP(I)                             RENAME
  225 CONTINUE
      WRITE(4,2020) ETOT, RETOT
      WRITE(4,202)
  202 FORMAT(///,8X,1HE,13X,5HTRANS,9X,9HREL ERROR)
      DO 230 I=1,35
      WRITE(4,210) E(I),TRANS(I),RTRANS(I)
  210 FORMAT(5X,1PE10.3,5X,1PE10.3,5X,0PF6.3)
  230 CONTINUE
      WRITE(4,2020) TTOT, RTTOT
      ABSORB = ABSORB/NPS
      WRG=WRG/NPP
      WRL=WRL/NPP
      WCP=WCP/NPP
      WCO=WCO/NPP
      WRITE(4,221) ABSORB, CUTOFF
  221 FORMAT(////,9HABSORB = ,1PE10.3,5X,9HCUTOFF = ,I5)
      WRITE(4,3728) NS,NR
 3728 FORMAT(28H TRACKS CREATED BY SPLITTING,I8,
     1 24H TRACKS LOST TO ROULETTE,I8)
      WRITE(4,3729) WRG,WRL
 3729 FORMAT(27H WEIGHT CREATED BY ROULEITE,1PE11.4,
     1 24H WEIGHT LOST TO ROULETTE,1PE11.4)
      WRITE(4,3730) NCO
 3730 FORMAT(29H TRACKS LOST TO WEIGHT CUTOFF,I8)
      WRITE(4,3731) WCP,WCO
 3731 FORMAT(32H WEIGHT CREATED BY WEIGHT CUTOFF,1PE11.4,
     1 29H WEIGHT LOST TO WEIGHT CUTOFF,1PE11.4)
      WRITE(4,2021) TEND
 2021 FORMAT(///, 13HTOTAL TIME = ,1PE10.3, 8H SECONDS)
      STOP
      END                                                               p14
```

```
      SUBROUTINE TRACK                                                  p17
      CALCULATE ALL INTERSECTIONS WITH ALL THREE SURFACES               p17

            COMMON X, Y, Z, U, V, W, ERG, WT, IA, NP                          BUGFIX

                              d05

            COMMON UOLD, VOLD, WOLD, CL, CL2, CRAD2, DLS, JA                  BUGFIX

                                    d03

            COMMON /NEW/ KRN                                            d13 BUGFIX

            DLSS = 1.0E10                                               p17
            IF(IA.EQ.2) GO TO 19                                        :
            DO 300 J=1,3                                                :
            D1 = -1.0                                                   :
            GO TO (55,160,50),J                                         :
         50 IF(V.EQ.0.) GO TO 300                                       :
            D1 = (CL-Y)/V                                               :
            GO TO 280                                                   :
         55 IF(V.EQ.0.) GO TO 300                                       :
            D1 = -Y/V                                                   :
            GO TO 280                                                   :
        160 T1 = U**2 + W**2                                            :
            IF(T1.EQ.0.) GO TO 300                                      :
            A1 = (X*U + Z*W)/T1                                         :
            B1 = (X**2 + Z**2 - CRAD2)/T1                               :
            T1 = A1**2 - B1                                             :
            IF(T1.LT.0.) GO TO 300                                      :
            T2 = SQRT(T1)                                               :
            D1 = -A1 + T2                                               :
            D2 = -A1 - T2                                               :
      C     IF(J.EQ.JA) D2=D1=-2.*A1                                  : OLD
            IF(J.EQ.JA) THEN                                          : NEW
               D1=-2.*A1                                              : NEW
               D2=-2.*A1                                              : NEW
            END IF                                                    : NEW
            GO TO 290                                                   :
        280 D2 = -D1                                                    :
        290 IF(D1.LE.0.) GO TO 300                                      :
            IF(D2.GT.0.) D1=D2                                          :
            IF(D1.GE.DLSS) GO TO 300                                    :
            JAS = J                                                     :
            DLSS = D1                                                   :
        300 CONTINUE                                                    :
            DLS = DLSS+1.0E-10                                          :
            JA = JAS                                                    :
            RETURN                                                      :
         19 DO 301 J=2,4                                                :
            D1 = -1.0                                                   :
            GO TO (56,161,51,56),J                                      :
         51 IF(V.EQ.0.) GO TO 301                                       :
            D1 = (CL-Y)/V                                               :
            GO TO 281                                                   :
         56 IF(V.EQ.0.) GO TO 301                                       p17
```

```
      D1 = (CL2-Y)/V                                                    p17
      GO TO 281
  161 T1 = U**2 + W**2
      IF(T1.EQ.0.) GO TO 301
      A1 = (X*U + Z*W)/T1
      B1 = (X**2 + Z**2 - CRAD2)/T1
      T1 = A1**2 - B1
      IF(T1.LT.0.) GO TO 301
      T2 = SQRT(T1)
      D1 = -A1 + T2
      D2 = -A1 - T2
C     IF (J.EQ.JA) D2=D1=-2.*A1                                         OLD
      IF (J.EQ.JA) THEN                                                 NEW
          D1=-2.*A1                                                     NEW
          D2=-2.*A1                                                     NEW
      END IF                                                            NEW
      GO TO 291
  281 D2 = -D1
  291 IF(D1.LE.0.) GO TO 301
      IF(D2.GT.0.) D1=D2
      IF(D1.GE.DLSS) GO TO 301
      JAS = J
      DLSS = D1
  301 CONTINUE
      DLS = DLSS+1.0E-10
      JA = JAS
      RETURN
      END

SUBROUTINE KLEIN(T1,T4)
SAMPLE FROM KLEIN-NISHINA USING INVERSE FIT
T1=ENERGY IN, T4=ENERGY OUT, IN UNITS OF THE REST MASS
C         OF AN ELECTRON.                                               p17
C
      COMMON /NEW/ KRN                                                  d13 BUGFIX

      RN=RANF(KRN)                                                      p17
      T2=1./T1
      T4=2.*T1+1.
      T5=1./T4
      T6=ALOG(T4)
      T3=2.*T1*(1.+T1)*T5**2+4.*T2+(1.-2.*T2*(1.+T2))*T6
      IF(T1.LE.1.16666667)GO TO 20
      T7=1.65898+T2*(.62537*T2-1.00796)
      T3=T7/T3
      IF(RN.LE.T3)GO TO 10
      T4=(T6-1.20397)/(1.-T3)
      T7=.3*EXP(T4*(T3-RN))
      GO TO 40
   10 T4=T7/(3.63333+T2*(5.44444*T2-4.66667))
      T7=.5*T7
      T2=RN/T3
      T3=2.1
      T5=1.4
      GO TO 30
   20 T4=T3/(T4+T5)                                                     p17
```

```
      T7= 5*T3                                                          p17
      T2=RN
      T5=1.-T5
      T3=3.*T5
      T5=2.*T5
   30 T7=1.+T2*(T2*(2.*T7+T4-T3+T2*(T5-T7-T4))-T7)
   40 T4=T7*T1
      RETURN
      END

      SUBROUTINE ISOS
C        SAMPLE A DIRECTION U,V,W ISOTROPICALLY.                        p17
C
      COMMON X,Y,Z,U,V,W,ERG,WT,IA,NP                                   BUGFIX

                        d05

      COMMON UOLD,VOLD,WOLD,CL,CL2,CRAD2,DLS,JA                         BUGFIX

                                  d03

      COMMON /NEW/ KRN                                                  d13 BUGFIX

   10 T1=2.*RANF(KRN)-1.                                                p17
      T2=2.*RANF(KRN)-1.
      RSQ=T1**2+T2**2
      IF(RSQ.GT.1.0)GO TO 10
      U=2.*RSQ-1.
      T3=SQRT((1.-U**2)/RSQ)
      V=T1*T3
      W=T2*T3
      RETURN
      END

      SUBROUTINE ROTAS(C)
C        ROTATE UOLD,VOLD,WOLD TO U,V,W THROUGH A POLAR
C        ANGLE WHOSE COSINE IS C, AND THROUGH AN AZIMUTHAL
C        ANGLE SAMPLED UNIFORMLY.                                       p17
C
      COMMON X,Y,Z,U,V,W,ERG,WT,IA,NP                                   BUGFIX

                        d05

      COMMON UOLD,VOLD,WOLD,CL,CL2,CRAD2,DLS,JA                         BUGFIX

                                  d03

      COMMON /NEW/ KRN                                                  d13 BUGFIX

   10 T1=2.*RANF(KRN)-1.                                                p17
      T2=2.*RANF(KRN)-1.
      R=T1**2+T2**2
      IF(R.GT.1.0)GO TO 10
      R=SQRT((1.-C**2)/R)
      T1=T1*R
      T2=T2*R                                                           p17
```

```
561         IF(ABS(WOLD).GT.999999)GO TO 30                          p17
562         S=SQRT(1.-WOLD**2)                                        !
563         U=UOLD*C+(T1*UOLD*WOLD-T2*VOLD)/S                         !
564         V=VOLD*C+(T1*VOLD*WOLD+T2*UOLD)/S                         !
565         W=WOLD*C-T1*S                                             !
566         RETURN                                                    !
567      30 U=I1                                                      !
568         V=T2                                                      !
569         W=WOLD*C                                                  !
570         RETURN                                                    !
571         END                                                    p17
572
573         SUBROUTINE SECOND (T)                                        NEW
574         T=0                                                          NEW
575         RETURN                                                       NEW
576         END                                                          NEW
577
578         REAL FUNCTION RAND0(KERN)                              p16 ALG CH
579         KERN = MOD(1+7421*KERN, 131072)                         !  ALG CH
580         RAND0 = FLOAT(KERN)/131072.                             !  ALG CH
581         RETURN                                                  !  ALG CH
582         END                                                    p16 ALG CH
583
584         REAL FUNCTION RANF(KERN)                               p17 ALG CH
585         KERN = MOD(1+9621*KERN, 131072)                         !  ALG CH
586         RANF = FLOAT(KERN)/131072.                              !  ALG CH
587         RETURN                                                  !  ALG CH
588         END                                                    p17 ALG CH
```

APPENDIX D

[lgdf]

LGDF Parallel GAMTEB

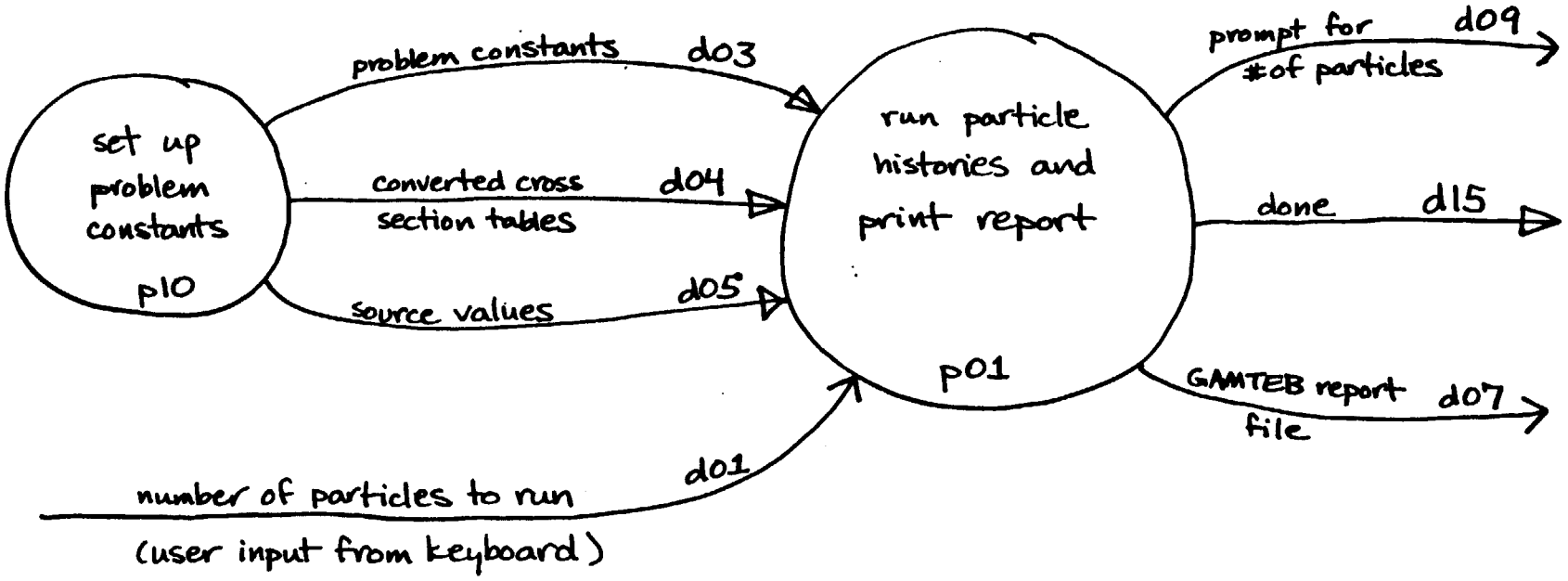
1. Project File - used to control the macro expansion process
2. LGDF Network Diagrams - express data-process dependencies
3. Wirelist - machine-readable form of LGDF network diagrams
4. Data Dictionary - description of data items associated with each data path
5. Data Declarations - FORTRAN fragments used to construct labeled COMMONs
6. Process Definitions - LGDF macro form

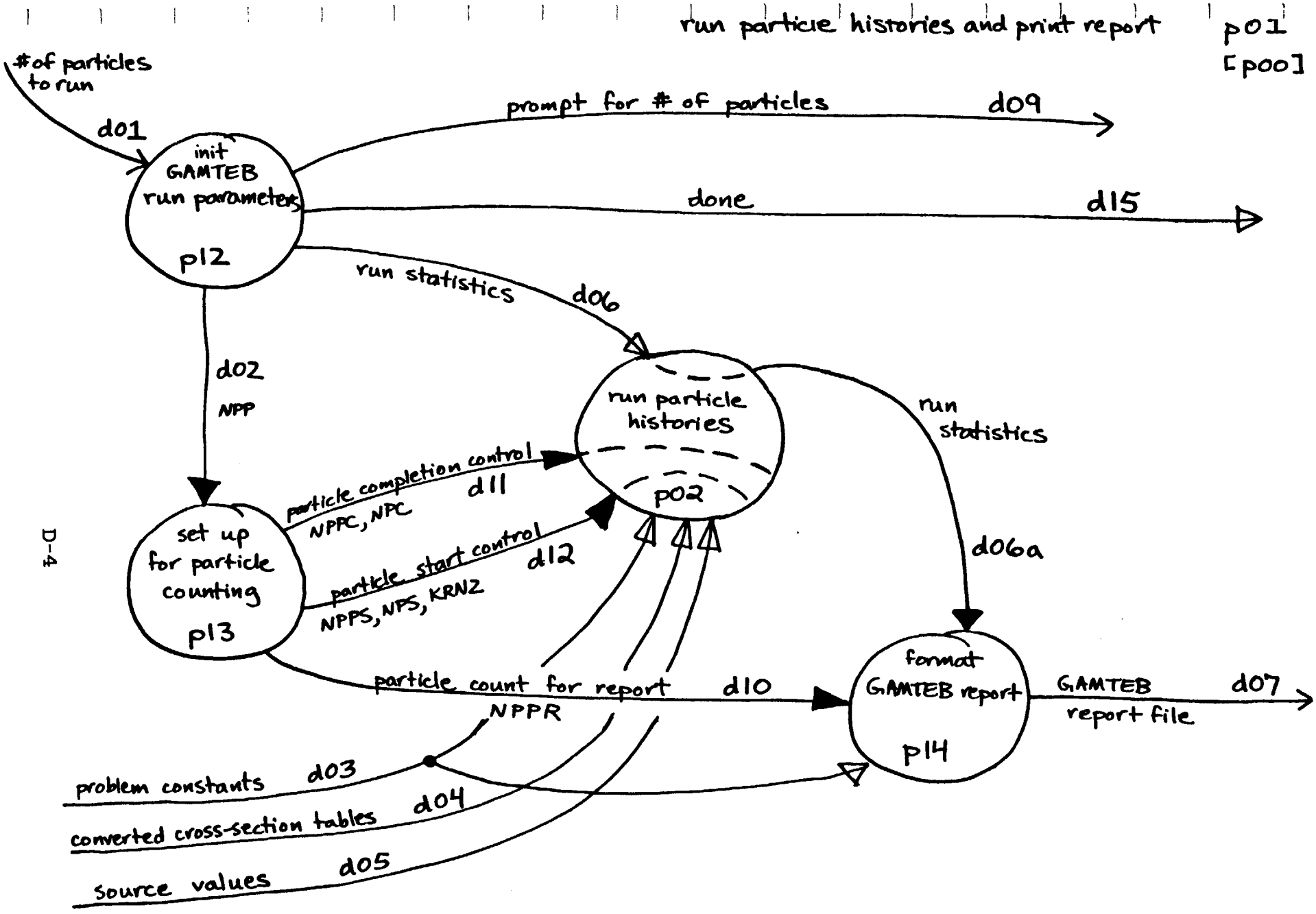
Apr 17 14:01 1985 Project Page 1

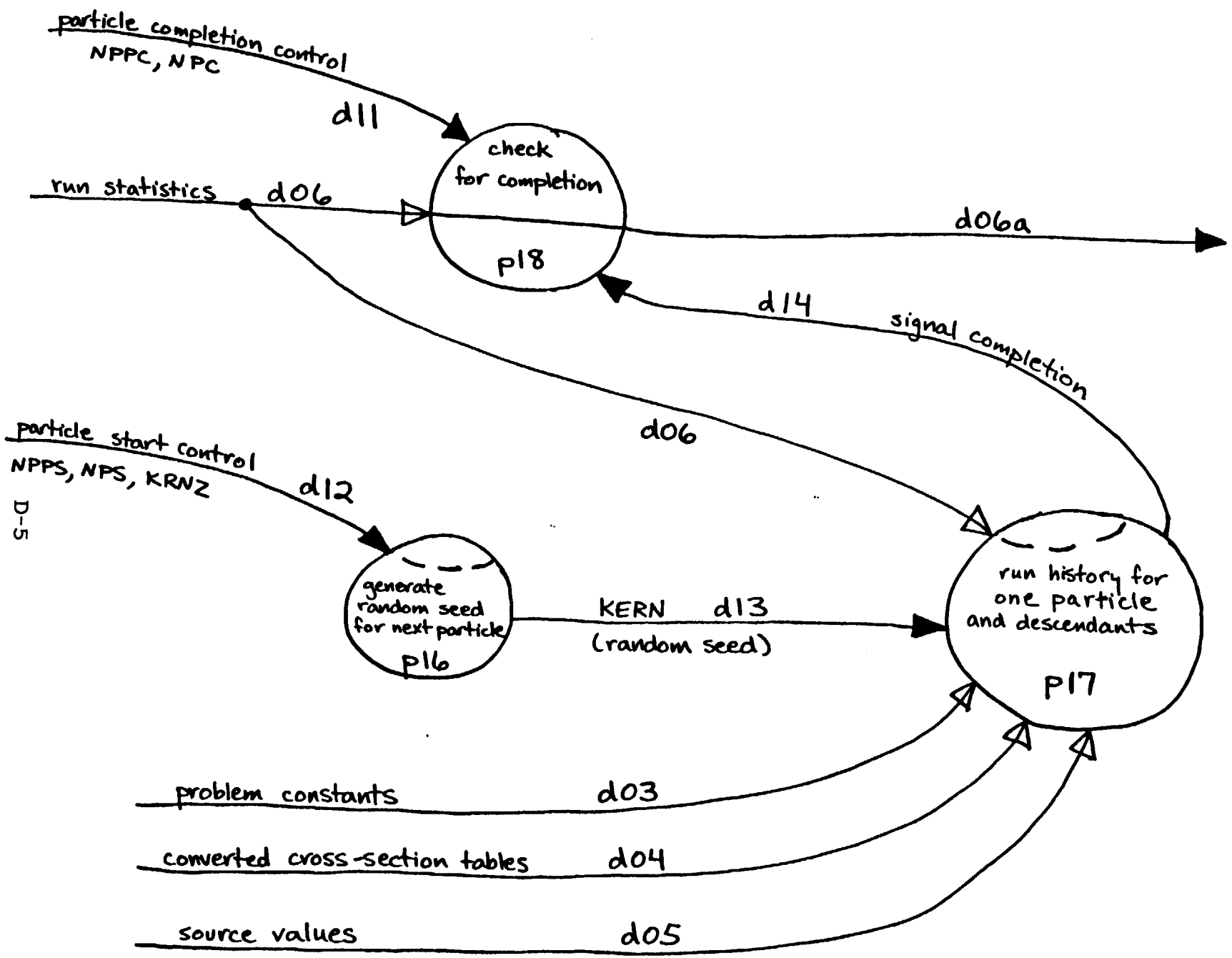
LGDF Project File:

```
  project_title_(Scalar Monte Carlo Transport Code)
   program_name_(GAMTEB)
        machine_(VAX)
       language_(FORTRAN)
       wirelist_(Wirelist)
 data_directory_(.)
       noptrace_()
       nogtrace_()
```

GAMTEB p00
[top]




run particle histories and print report
#of particles to run
d01
init GAMTEB run parameters
p12
prompt for # of particles
d09
done
d15
run statistics
d06
d02
NPP
run particle histories
p02
run statistics
d06a
particle completion control
NPPC, NPC
d11
particle start control
NPPS, NPS, KRNZ
d12
set up for particle counting
p13
particle count for report
NPPR
d10
format GAMTEB report
p14
GAMTEB report file
d07
problem constants
d03
converted cross-section tables
d04
source values
d05


particle completion control
NPPC, NPC
d11
check for completion
p18
run statistics
d06
d06a
d14
signal completion
d06
particle start control
NPPS, NPS, KRNZ
d12
generate random seed for next particle
p16
KERN d13
(random seed)
run history for one particle and descendants
p17
problem constants
d03
converted cross-section tables
d04
source values
d05

```
*** DEFINE DATA PATHS

***     tag  short name     descriptive name

_defdp(d01, "user input", "No. of Particles to run")
_defdp(d02, "NPP",        "No. of Particles to run")
_defdp(d03, "constants",  "problem constants")
_defdp(d04, "cs tables",  "cross section tables")
_defdp(d05, "src values", "source values")
_defdp(d06, "run stats",  "run statistics")
_defdp(d07, "report",     "GAMTEB report file")
_defdp(d09, "prompts",    "prompt for no. of particles")
_defdp(d10, "NPPR",       "particle count for report")
_defdp(d11, "NPPC;NPC",   "particle completion control")
_defdp(d12, "PSC",        "particle start control")
_defdp(d13, "KERN",       "random seed")
_defdp(d14, "signal",     "particle history completion signal")
_defdp(d15, "done",       "run completion interlock")

*** DEFINE PROGRAMS & NETWORKS

***     tag  short name     descriptive name

_defpn(p00, "GAMTEB",     "Scalar Monte Carlo Transport Code")
_defpn(p01, "RUNPR",      "run particle histories and print report")
_defpn(p02, "RUNPH",      "run particle history")
_defpn(p10, "SETUP",      "set up problem constants")
_defpn(p12, "PARAM",      "init GAMTEB run parameters")
_defpn(p13, "PCOUNT",     "setup for particle counting")
_defpn(p14, "REPORT",     "format GAMTEB report")
_defpn(p16, "GENRAN",     "generate random seed for next particle")
_defpn(p17, "RNHIST",     "run history for one particle and offspring")
_defpn(p18, "CKCOMP",     "check for run completion")

*** WIRELIST

*****************************
_net(top,[])
   _in(d01,SE)
   _sys(p00,[top])
      _in(d01,SE)
      _out(d09,SE)
      _waitout(d15,NC)
      _out(d07,SE)
   _endsys(p00,[top])
_endnet(top,[])
*****************************
_net(p00,[top])
   _in(d01,SE)
   _prog(p10,[p00])
      _out(d03,NC)
      _out(d04,NC)
      _out(d05,NC)
   _endprog(p10,[p00])
   _sys(p01,[p00])
      _in(d03,NC)
```

```
        _in(d04,NC)
        _in(d05,NC)
        _in(d01,SE)
        _out(d09,SE)
        _out(d15,NC)
        _out(d07,SE)
    _endsys(p01,[p00])
_endnet(p00,[top])
*****************************
_net(p01,[p00])
    _in(d01,SE)
    _in(d03,NC)
    _in(d04,NC)
    _in(d05,NC)
    _prog(p12,[p01])
        _in(d01,SE)
        _out(d09,SE)
        _out(d15,NC)
        _out(d06,NC)
        _out(d02,CL)
    _endprog(p12,[p01])
    _prog(p13,[p01])
        _in(d02,CL)
        _out(d10,CL)
        _out(d11,CL)
        _out(d12,CL)
    _endprog(p13,[p01])
    _sys(p02,[p01])
        _in(d06,NC,UP)
        _in(d11,CL,UP)
        _in(d12,CL,UP)
        _in(d03,NC)
        _in(d04,NC)
        _in(d05,NC)
            _shared(d06,d06a,NC,UP)
        _out(d06a,CL)
    _endsys(p02,[p01])
    _prog(p14,[p01])
        _in(d06a,CL)
        _in(d10,CL)
        _in(d03,NC)
        _out(d07,SE)
    _endprog(p14,[p01])
_endnet(p01,[p00])
*****************************
_net(p02,[p01])
    _in(d11,CL)
    _in(d06,NC)
    _in(d12,CL,UP)
    _in(d03,NC)
    _in(d04,NC)
    _in(d05,NC)
    _prog(p16,[p02])
        _in(d12,CL,UP)
        _out(d13,CL)
    _endprog(p16,[p02])
```

```
    _prog(p17,[p02],*30)
        _in(d06,NC,UP)
        _in(d13,CL)
        _in(d03,NC)
        _in(d04,NC)
        _in(d05,NC)
        _out(d14,CL)
    _endprog(p17,[p02])
    _prog(p18,[p02])
        _in(d06,NC,R0)
        _in(d11,CL)
        _in(d14,CL)
            _shared(d06,d06a,NC,R0)
        _out(d06a,CL)
    _endprog(p18,[p02])
_endnet(p02,[p01])
*******************************
```

```
d01
  IDO1 - number of particles to run (input from user)

d02
  NPP - number of particles

d03 - problem constants
  E(35) - energy constant table
  RHO - constant
  CL - cylinder length
  CL2 - cylinder length + 10
  CRAD - cylinder radius
  CRAD2 - cylinder radius squared (CRD2)
  WCP1 - weight cutoff 1
  WCP2 - weight cutoff 2
  EC - energy cutoff
  FIM(2) - flip

d04 - converted cross section tables
  EL(35) - log of energy constant table
  XC(35)
  XPP(35)
  XPE(35)

d05 - source values
  GIERG - energy
  GIWT - mass
  GIU - direction
  GIV      "
  GIW      "
  GIX - position
  GIY      "
  GIZ      "
  IGIA

d06 - run statistics
  IGNCOL - number of collisions
  IGNCO - tracks lost to weight cutoff
  GWCO - weight lost to weight cutoff
  GWCP - weight created by weight cutoff
  GWRL - weight lost in russian roulette
  IGNR - tracks lost to roulette
  GWRG - weight created by roulette
  IGCUTF - no. of particles cut off when energy .LE. EC:d4b
  GABSOR - weight absorbed
  IGNS - no. of tracks created by splitting
  GTRANS(35) - transmitted weight total by energy level
  GESCAP(35) - escaped weight total by energy level
  GBSCAT(35) - back-scattered weight total by energy level
  GTRNS2(35) - sum of squares of GTRANS
  GESCP2(35) - sum of squares of GESCAP
  GBSCT2(35) - sum of squares of GBSCAT
  GTRNSI(35) -
  GBSCTI(35) -
  GESCPI(35) -
  GBTOTI - back-scattered weight total
```

```
  GETOTI - escaped weight total
  GTTOTI - transmitted weight total
  GBTOT2 - sum of squares of BTOTI
  GTTOT2 - sum of squares of TTOTI
  GETOT2 - sum of squares of ETOTI

d07
  gamteb report

d08
  [not assigned]

d09
  prompt for # of particles

d10
  NPPR - particle count for report

d11 - particle completion control
  NPPC - no. of particle history summary statistics (d14) expected
         (= NPP (d02))
  NPC - particle history completion counter

d12 -   particle start control
  NPPS - no. of particles to start
         (= NPP (d02))
  NPS - particle history start counter
  KRN2 - second random number seed

d13
  KERN - first random number seed    ($KRN)

d14
  NCOL - number of collisions
  NCO - tracks lost to weight cutoff
  WCO - weight lost to weight cutoff
  WCP - weight created by weight cutoff
  WRL - weight lost in russian roulette
  NR - tracks lost to roulette
  WRG - weight created by roulette
  NPS - particle counter
  CUTOFF - no. of particles cut off when energy .LE. EC:d4b
  ABSORB - weight absorbed
  NS - no. of tracks created by splitting
  TRANS -
  ESCAPE -
  BSCAT -
  TRANS2 -
  ESCAP2 -
  BSCAT2 -
  TRANSI -
  BSCATI -
  ESCAPI -
  BTOTI - back-scattered weight total
  ETOTI - escaped weight total
  TTOTI - transmitted weight total
```

BTOT2 -
TTOT2 -
ETOT2 -

```
::::::::::::::
dO1
::::::::::::::

IDO1
      INTEGER IDO1


::::::::::::::
dO2
::::::::::::::

NPP
      INTEGER NPP


::::::::::::::
dO3
::::::::::::::

E,RHO,CL,CL2,CRAD,CRAD2,WCP1,WCP2,EC,FIM
      REAL E(35),RHO,CL,CL2,CRAD,CRAD2,WCP1,WCP2,EC,FIM(2)


::::::::::::::
dO4
::::::::::::::

EL,XC,XPP,XPE
      REAL EL(35),XC(35),XPP(35),XPE(35)
```

```
::::::::::::::
d05
::::::::::::::

GIERG,GIWT,GIU,GIV,GIW,GIX,GIY,GIZ,IGIA

      REAL GIERG,GIWT,GIU,GIV,GIW,GIX,GIY,GIZ

      INTEGER IGIA




::::::::::::::
d06
::::::::::::::

IGNCOL,IGNCO,GWCO,GWCP,GWRL,IGNR,GWRG,IGCUTF,
     +   GABSOR,IGNS,
     +   TRANS,ESCAPE,BSCAT,TRANS2,ESCAP2,BSCAT2,
     +   BTOT2,TTOT2,ETOT2,ETOT,BTOT,TTOT

      INTEGER IGNCOL,IGNCO,IGNR,IGCUTF,IGNS

      REAL GWCO,GWCP,GWRL,GWRG,GABSOR,
     +   TRANS(35),ESCAPE(35),BSCAT(35),
     +   TRANS2(35),ESCAP2(35),BSCAT2(35),
     +   BTOT2,TTOT2,ETOT2,ETOT,BTOT,TTOT



::::::::::::::
d07
::::::::::::::

IDO7

      INTEGER IDO7
```

```
::::::::::::::
d09
::::::::::::::

CL,CL2,CRAD,CRAD2,WCP1,WCP2,EC
      REAL CL,CL2,CRAD,CRAD2,WCP1,WCP2,EC


::::::::::::::
d10
::::::::::::::

NPPR
      INTEGER NPPR


::::::::::::::
d11
::::::::::::::

NPC,NPPC
      INTEGER NPC,NPPC


::::::::::::::
d12
::::::::::::::

NPPS,NPS,KRN2
      INTEGER NPPS,NPS,KRN2
```

```
::::::::::::::
d13
::::::::::::::

KERN
      INTEGER KERN


::::::::::::::
d14
::::::::::::::

ID14
      INTEGER ID14


::::::::::::::
d15
::::::::::::::

IDO15
      INTEGER IDO15
```

```
1  top_(p00)
2
```

```
      program_(p10,tp001)
      begin_
            CL=20.0
            CL2=CL+10.
            CRAD=1.0
            CRAD2=CRAD**2
            WCP1=.5
            WCP2=.25
            EC = .001
            FIM(1)=1.0                                                    STYLE
            FIM(2)=2.0
            set_(d03)
C
C           CONVERT CROSS-SECTION UNITS TO BE PER CM.
C
            DO 1 I=1,35
            XC(I)=ALOG( XC(I)*RHO )
            IF(XPP(I).EQ.0.) XPP(I)=1.0E-37                               NEW
            IF(XPE(I).EQ.0.) XPE(I)=1.0E-37                               NEW
            XPP(I)=ALOG( XPP(I)*RHO )
            XPE(I)=ALOG( XPE(I)*RHO )
            EL(I)=ALOG(E(I))
          1 CONTINUE
            set_(d04)
C
C           SET SOURCE VALUES
C
            GIERG = 6.0
            GIWT = 1.0
            GIU = 0.0
            GIV = 1.0
            GIW = 0.0
            GIX = 0.0
            GIY = .000001
            GIZ = 0.0
            IGIA=1
            set_(d05)
            suspend_
      end_(p10)
            BLOCK DATA
      _d(d03)
      _d(d04)
            DATA RHO/2.22/
            DATA (E(I),I=1,35)/ .001, .0015, .002, .003, .004, .005,
           1      .006, .008, .01, .015, .02, .03, .04, .05, .06, .08,
           2      .1, .15, .2, .3, .4, .5, .6, .8, 1., 1.5, 2., 3., 4., 5.,
           3      6., 8., 10., 15., 20./
            DATA (XC(I),I=1,35)/.0150,.0296,.0451,.0717,.0913
           1,.105,.115,.128,.137,.152
           2,.160,.165,.165,.163,.160
           3,.153,.146,.133,.122,.106,.0953,.0867,.0802,.0707,.0637
           4,.0516,.044,.0346,.0289,.0250,.0221,.0181,.0154,.0114,.00913/
            DATA (XPP(I),I=1,35)/ 0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,
           1      0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,.0000792,
           2      .000316, .000923, .00153, .00208, .00256, .00343,
           3      .00414, .00547, .00652/
```

```
      DATA (XPE(I),I=1,35)/ 2010., 632., 280., 87.7, 37.3, 18.9,
     1     10.4, 4.01, 1.91, .489, .192, .0491, .0186, .00887,
     2     .00481, .00179, .000862, .000234, .0000918,
     4  0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0./
      END
```

```
 1  program_(p12,[p01])
 2  begin_
 3  C          INITIALIZE PROBLEM INPUT
 4  C
 5          lockio_
 6          WRITE(6,*) 'p12: ENTER NO. OF PARTICLES TO RUN'                  UPGRADE
 7  C       NPP=500000                                                       OLD
 8          READ(5,*) NPP                                                    UPGRADE
 9          unlockio_
10          IF (NPP.EQ.0) THEN
11              set_(d15)
12              suspend_
13          ELSE
14              set_(d02)
15          ENDIF
16  C
17          IGNCOL=0
18          IGNCO=0
19          GWCO=0
20          GWCP=0
21          GWRL=0.
22          GWRG=0.
23          GABSOR=0.
24          IGNR=0
25          IGCUTF= 0
26          IGNS=0                                                           BUGFIX
27          DO 5 I = 1,35
28             TRANS2(I)= 0.0
29             BSCAT2(I)= 0.0
30             ESCAP2(I)= 0.0                                                RENAME
31             TRANS(I) = 0.0
32             BSCAT(I) = 0.0
33             ESCAPE(I) = 0.0
34        5 CONTINUE                                                         STYLE
35          BTOT=0.0
36          BTOT2=0.0
37          TTOT=0.0
38          TTOT2=0.0
39          ETOT=0.0
40          ETOT2=0.0
41          set_(d06)
42          suspend_
43  end_(p12)
```

```
 1  program_(p13,[p01])
 2  begin_
 3  C
 4  C      KRN = 123454321                                              OLD
 5  C -- CHANGE TO KRN2 DUE TO ADDITION OF SECOND R.N. GENERATOR        ALG CH
 6  C -- SET UP PARTICLE START CONTROL                                  NEW
 7         KRN2 = 123                                                   ALG CH
 8         NPPS=NPP                                                     NEW
 9         NPS=0
10         set_(d12)
11  C -- SET UP PARTICLE COMPLETION CONTROL
12         NPC=0                                                        NEW
13         NPPC=NPP                                                     NEW
14         set_(d11)
15  C -- MAKE COPY OF NPP FOR REPORT PROCESS
16         NPPR=NPP
17         set_(d10)
18         clear_(d02)
19         suspend_
20  end_(p13)
21
```

```
      1  program_(p14,[p01])
      2        REAL RTRANS(35),RBSCAT(35),RESCAP(35)                    RENAME
      3  begin_
      4  C
      5  C          PRINT OUTPUT
      6  C
      7  C -- NPP AND NPS ARE USED INTERCHANGEABLY IN THE ORIGINAL CODE  NEW
      8  C 140 NPS = NPS - 1                                             OLD
      9    140 NPS=NPPR                                                  NEW
     10  C -- FIX UP NAME ALIAS FOR NPP                                  NEW
     11        NPP=NPPR                                                  NEW
     12  C -- GET EXCLUSIVE ACCESS TO I/O FOR PRINT FILE                 HEP
     13        lockio_
     14        WRITE(4,7634) IGNCOL
     15   7634 FORMAT(5H NCOL,I10)
     16        WRITE(4,1401)
     17   1401 FORMAT(7HSCALERT,/)
     18        WRITE(4,150) NPS
     19    150 FORMAT(6HNPS =,I6)
     20        WRITE(4,200)
     21    200 FORMAT(///,8X,1HE,13X,5HBSCAT,9X,9HREL ERROR)
     22        DO 220 I=1,35
     23        RNPS= NPS
     24        TRANS(I) = TRANS(I)/RNPS
     25        BSCAT(I) = BSCAT(I)/RNPS
     26        ESCAPE(I) = ESCAPE(I)/RNPS
     27        TRANS2(I)=TRANS2(I)/RNPS
     28        BSCAT2(I)=BSCAT2(I)/RNPS
     29        ESCAP2(I)=ESCAP2(I)/RNPS                                 RENAME
     30        IF(TRANS(I).NE.0.0)GO TO 203
     31        RTRANS(I)= 0.0
     32        GO TO 204
     33    203 RTRANS(I)= SQRT((TRANS2(I)-TRANS(I)**2)/ RNPS)
     34        RTRANS(I)= RTRANS(I)/TRANS(I)
     35    204 IF(BSCAT(I).NE.0.0)GO TO 205
     36        RBSCAT(I)= 0.0
     37        GO TO 206
     38    205 RBSCAT(I)= SQRT((BSCAT2(I)-BSCAT(I)**2)/ RNPS)
     39        RBSCAT(I)= RBSCAT(I)/ BSCAT(I)
     40    206 IF(ESCAPE(I).NE.0.0)GO TO 207
     41        RESCAP(I)= 0.0                                           RENAME
     42        GO TO 209
     43    207 RESCAP(I)= SQRT((ESCAP2(I)-ESCAPE(I)**2)/ RNPS)          RENAME
     44        RESCAP(I)= RESCAP(I)/ ESCAPE(I)                          RENAME
     45    209 WRITE(4,210) E(I),BSCAT(I),RBSCAT(I)
     46    220 CONTINUE
     47        TTOT = TTOT/RNPS
     48        TTOT2 = TTOT2/RNPS
     49        BTOT = BTOT/RNPS
     50        BTOT2 = BTOT2/RNPS
     51        ETOT = ETOT/RNPS
     52        ETOT2 = ETOT2/RNPS
     53        IF(TTOT.NE.0.0) GO TO 2000
     54        RTTOT = 0.0
     55        GO TO 2001
     56   2000 RTTOT = SQRT((TTOT2 - TTOT**2)/RNPS)
```

```
      RTTOT = RTTOT/TTOT
 2001 IF(BTOT.NE.0.0) GO TO 2002
      RBTOT = 0.0
      GO TO 2003
 2002 RBTOT = SQRT((BTOT2 - BTOT**2)/RNPS)
      RBTOT = RBTOT/BTOT
 2003 IF(ETOT.NE.0.0) GO TO 2004
      RETOT = 0.0
      GO TO 2005
 2004 RETOT = SQRT((ETOT2 - ETOT**2)/RNPS)
      RETOT = RETOT/ETOT
 2005 CONTINUE
      WRITE(4,2020) BTOT, RBTOT
 2020 FORMAT(/,6X,5HTOTAL,9X,1PE10.3,5X,0PF7.4)
      WRITE(4,201)
  201 FORMAT(///,8X,1HE,13X,6HESCAPE,8X,9HREL ERROR)
      DO 225 I=1,35
      WRITE(4,210) E(I),ESCAPE(I),RESCAP(I)                    RENAME
  225 CONTINUE
      WRITE(4,2020) ETOT, RETOT
      WRITE(4,202)
  202 FORMAT(///,8X,1HE,13X,5HTRANS,9X,9HREL ERROR)
      DO 230 I=1,35
      WRITE(4,210) E(I),TRANS(I),RTRANS(I)
  210 FORMAT(5X,1PE10.3,5X,1PE10.3,5X,0PF6.3)
  230 CONTINUE
      WRITE(4,2020) TTOT, RTTOT
      GABSOR = GABSOR/NPS
      GWRG=GWRG/NPP
      GWRL=GWRL/NPP
      GWCP=GWCP/NPP
      GWCO=GWCO/NPP
      WRITE(4,221) GABSOR, IGCUTF
  221 FORMAT(////,9HABSORB = ,1PE10.3,5X,9HCUTOFF = ,I5)
      WRITE(4,3728) IGNS,IGNR
 3728 FORMAT(28H TRACKS CREATED BY SPLITTING,I8,
     1 24H TRACKS LOST TO ROULETTE,I8)
      WRITE(4,3729) GWRG,GWRL
 3729 FORMAT(27H WEIGHT CREATED BY ROULETTE,1PE11.4,
     1 24H WEIGHT LOST TO ROULETTE,1PE11.4)
      WRITE(4,3730) IGNCO
 3730 FORMAT(29H TRACKS LOST TO WEIGHT CUTOFF,I8)
      WRITE(4,3731) GWCP,GWCO
 3731 FORMAT(32H WEIGHT CREATED BY WEIGHT CUTOFF,1PE11.4,
     1 29H WEIGHT LOST TO WEIGHT CUTOFF,1PE11.4)
      WRITE(4,2021) TEND
 2021 FORMAT(///, 13HTOTAL TIME = ,1PE10.3, 8H SECONDS)
C -- RETURN ACCESS TO I/O
      unlockio_
      clear_(d06)
      clear_(d10)
      suspend_
end_(p14)
```

```
 1  program_(p16,[p02])
 2  begin_
 3  C    START A HISTORY
 4      10 NPS = NPS + 1
 5         IF(NPS.GT.NPPS) THEN
 6  C      GO TO 140                                                    OLD
 7             clear_(d12)
 8         ELSE
 9  C         GENERATE NEW RANDOM SEED                                  ALG CH
10            XJUNK = RANDO(KRN2)                                       ALG CH
11            KERN = KRN2                                               ALG CH
12            set_(d13)
13         ENDIF
14         suspend_
15  end_(p16)
16
17         REAL FUNCTION RANDO(KERN)                                    ALG CH
18         KERN = MOD(1+7421*KERN,131072)                               ALG CH
19       4 FORMAT(1X,F12.8)                                             ALG CH
20         RANDO = FLOAT(KERN)/131072.                                  ALG CH
21         RETURN
22         END
```

```
      program_(p17,fp021)
C
C        SCALAR MONTE CARLO CODE TO TRANSPORT .001 TO 20.0 MEV
C        GAMMA RAYS IN A CARBON CYLINDER OF LENGTH CL, RADIUS CRAD
C
      REAL BANK(100,8),PBL(8)                                          BUGFIX
      INTEGER IBANK(100,2),IPBL(2)                                     BUGFIX
      EQUIVALENCE (PBL(1),X),(PBL(2),Y),(PBL(3),Z),                    BUGFIX
     +            (PBL(4),U),(PBL(5),V),(PBL(6),W),                    BUGFIX
     +            (PBL(7),ERG),(PBL(8),WT),                            BUGFIX
     +            (IPBL(1),IA),(IPBL(2),NP)                            BUGFIX
      REAL BSCATI(35),TRANSI(35),ESCAPI(35)                            RENAME
      REAL BTOTI,TTOTI,ETOTI
      INTEGER INBNK,NBANK
      INTEGER KRN                                                      ALG CH
      INTEGER CUTOFF
begin_
      aread_(d13)
C -- MAKE LOCAL COPY OF RANDOM SEED
      KRN=KERN
      aclear_(d13)
C -- MAKE LOCAL COPIES OF INITIAL SOURCE VALUES
      ERG=GIERG
      WT=GIWT
      U=GIU
      V=GIV
      W=GIW
      X=GIX
      Y=GIY
      Z=GIZ
      IA=IGIA
C
      NCOL=0
      NCO=0
      WCO=0
      WCP=0
      WRL=0.
      WRG=0.
      NR=0
      CUTOFF= 0
      ABSORB=0
      NS=0                                                             BUGFIX
      DO 5 I = 1,35
         BSCATI(I)=0.                                                  BUGFIX
         TRANSI(I)=0.                                                  BUGFIX
         ESCAPI(I)=0.                                                  BUGFIX
    5 CONTINUE                                                         STYLE
      BTOTI=0.                                                         BUGFIX
      TTOTI=0.                                                         BUGFIX
      ETOTI=0.                                                         BUGFIX
C     INBNK=0                                                          EXTRA
C -- INITIALIZE LOCAL PARTICLE BANK INDEXES
      NBANK=0
      INBNK=0
C
C
```

```
C         CALCULATE DISTANCE DLS TO NEXT SURFACE INTERSECTION
C         FOR ALL THREE SURFACES AND ALSO THE NUMBER JA OF THE
C         NEXT SURFACE INTERSECTED
C
   20 JA=0
      CALL TRACK(IA,JA,X,Y,Z,U,V,W,CL,CL2,CRAD2,DLS)
C
C         FIND ENERGY POINTER FOR CROSS SECTIONS AND TALLYS
C
      DO 30 IE = 1,35
         IF(ERG.GT.E(IE)) GO TO 30
         I = IE
         GO TO 31
   30 CONTINUE
C
C         INTERPOLATION TO GET CROSS SECTIONS AS F(ERG)
C
   31 F=(ALOG(ERG)-EL(I-1))/(EL(I)-EL(I-1))
      XSC=EXP( XC(I-1)+F*(XC(I)-XC(I-1)) )
      XSPP=EXP( XPP(I-1)+F*(XPP(I)-XPP(I-1)) )
      XSPE=EXP( XPE(I-1)+F*(XPE(I)-XPE(I-1)) )
      XST  = XSC + XSPP + XSPE
C
C         CALCULATE DISTANCE TO NEXT COLLISION
      S = -ALOG(RANF(KRN))/XST
C
C         SEE IF COLLISION IS STILL INSIDE CYLINDER
C         IF NOT, DO TALLYS; IF SO, DO COLLISION PHYSICS
      IF(S.LT.DLS) GO TO 60
      X=X+U*DLS
      Y=Y+V*DLS
      Z=Z+W*DLS
      GO TO(42,50,53,52) JA
   42 BSCATI(I) = BSCATI(I) + WT
      BTOTI = BTOTI + WT
      GO TO 11
   52 TRANSI(I) = TRANSI(I) + WT
      TTOTI = TTOTI + WT
      GO TO 11
   50 ESCAPI(I) = ESCAPI(I) + WT                                    RENAME
      ETOTI = ETOTI + WT
      GO TO 11
C  CROSS INTERNAL SURFACE  SPLIT OR ROULETTE
   53 IAP=IA
      IA=2-IA/2
      T1=FIM(IA)/FIM(IAP)
      IF(T1.GT.1.0) GO TO 57
C  RUSSIAN ROULETTE
      IF(T1.LT.RANF(KRN)) GO TO 58
      WTSAV=WT
      WT=WT/T1
      WRG=WRG+(WT-WTSAV)
      GO TO 20
C  KILLED IN RUSSIAN ROULETTE
   58 WRL=WRL+WT
      NR=NR+1
```

```
      GO TO 11
C  SPLITTING
 57 NP=T1-1.
      WT=WT/T1
      NS=NS+NP
      NBANK=NBANK+NP
      INBNK=INBNK+1
      DO 59 IX=1,8                                                    BUGFIX
 59 BANK(INBNK,IX)=PBL(IX)
      DO 61 IX=1,2                                                    BUGFIX
 61 IBANK(INBNK,IX)=IPBL(IX)                                          BUGFIX
      GO TO 20
C  CHECK BANK BEFORE STARTING NEW PARTICLE
 11 IF(NBANK.EQ.0) GO TO 234
      DO 521 IX=1,8
521 PBL(IX)=BANK(INBNK,IX)
      DO 522 IX=1,1
522 IPBL(IX)=IBANK(INBNK,IX)
      NBANK=NBANK-1
      IBANK(INBNK,2)=IBANK(INBNK,2)-1                                 BUGFIX
      IF(IBANK(INBNK,2).EQ.0) INBNK=INBNK-1
      GO TO 20
C
C        COLLISIONS
 60 JA = 0
      X=X+U*S
      Y=Y+V*S
      Z=Z+W*S
      NCOL=NCOL+1
C  SURVIVAL BIAS
      WTSAV=WT
      WT=WT*(1.-XSPE/XST)
      ABSORB=ABSORB+(WTSAV-WT)
      XSTSB=XST-XSPE
C WEIGHT CUTOFF
      IF(WT.GT.WCP2) GO TO 832
      IF(WT*FIM(IA).LT.RANF(KRN)*WCP1*FIM(1)) GO TO 642
      WTSAV=WT
      WT=WCP1*FIM(1)/FIM(IA)
      WCP=WCP+(WT-WTSAV)
832 CONTINUE
      IF(RANF(KRN).GE.XSC/XSTSB) GO TO 100
      T1 = 1.956917*ERG
C        GET NEW ENERGY T4 AND COMPTON SCATTERING ANGLE
      CALL KLEIN(T1,T4,KRN)                                           ARGS
      CSA = 1.+1./T1-1./T4
      T5 = .511008*T4
      IF(ABS(CSA).GT.1.) CSA=SIGN(1.,CSA)
      ERG = T5
C
C        SEE IF NEW ENERGY IS LESS THAN CUTOFF
      IF(ERG.GT.EC) GO TO 70
      CUTOFF = CUTOFF + 1
      GO TO 11
C        MAKE COMPTON ANGLE RELATIVE TO PROBLEM COORDINATE SYSTEM
 70 UOLD = U
```

```
      VOLD = V
      WOLD = W
      CALL ROTAS(CSA,KRN,U,V,W,UOLD,VOLD,WOLD)                    ARGS
      GO TO 20
C
C        PAIR PRODUCTION
  100 ERG = 0.511008
      WT = 2.*WT
C
C        CHECK ENERGY CUTOFF
      IF(ERG.GT.EC) GO TO 110
      CUTOFF = CUTOFF + 1
      GO TO 11
C
C        ISOTROPIC EMISSION IN LAB SYSTEM
  110 CALL ISOS(U,V,W,KRN)                                        ARGS
      GO TO 20
C
C        PHOTOELECTRIC ABSORPTION
C NOW HANDLED BY SURVIVAL BIASING
C 130 ABSORB = ABSORB + WT
C     GO TO 11
C  TERMINATE PARTICLE TO WEIGHT CUTOFF
  642 WCO=WCO+WT
      NCO=NCO+1
      GO TO 11
C -- GET EXCLUSIVE UPDATE ACCESS TO RUN STATISTICS
  234 aread_(d06)
      IGNCOL=IGNCOL+NCOL
      IGNCO=IGNCO+NCO
      GWCO=GWCO+WCO
      GWCP=GWCP+WCP
      GWRL=GWRL+WRL
      GWRG=GWRG+WRG
      IGNR=IGNR+NR
      IGCUTF=IGCUTF+CUTOFF
      GABSOR=GABSOR+ABSORB
      IGNS=IGNS+NS
      DO 829 I=1,35
      BSCAT(I)=BSCAT(I)+BSCATI(I)
      BSCAT2(I)=BSCAT2(I)+BSCATI(I)**2
      TRANS(I)=TRANS(I)+TRANSI(I)
      TRANS2(I)=TRANS2(I)+TRANSI(I)**2
      ESCAPE(I)=ESCAPE(I)+ESCAPI(I)                               RENAME
      ESCAP2(I)=ESCAP2(I)+ESCAPI(I)**2                            RENAME
  829 CONTINUE
      BTOT=BTOT+BTOTI
      TTOT=TTOT+TTOTI
      ETOT=ETOT+ETOTI
      BTOT2=BTOT2+BTOTI**2
      TTOT2=TTOT2+TTOTI**2
      ETOT2=ETOT2+ETOTI**2
C -- GIVE BACK UPDATE ACCESS TO RUN STATISTICS
      unread_(d06)
C -- SIGNAL COMPLETION OF THIS PARTICLE HISTORY
      set_(d14)
```

```
C       GO TO 10                                                        OLD
        suspend_
end_(p17)
        SUBROUTINE TRACK(IA,JA,X,Y,Z,U,V,W,CL,CL2,CRAD2,DLS)            ARGS
C          CALCULATE ALL INTERSECTIONS WITH ALL THREE SURFACES
        DLSS = 1.0E10
        IF(IA.EQ.2) GO TO 19
        DO 300 J=1,3
        D1 = -1.0
        GO TO (55,160,50),J
   50   IF(V.EQ.0.) GO TO 300
        D1 = (CL-Y)/V
        GO TO 280
   55   IF(V.EQ.0.) GO TO 300
        D1 = -Y/V
        GO TO 280
  160   T1 = U**2 + W**2
        IF(T1.EQ.0.) GO TO 300
        A1 = (X*U + Z*W)/T1
        B1 = (X**2 + Z**2 - CRAD2)/T1
        T1 = A1**2 - B1
        IF(T1.LT.0.) GO TO 300
        T2 = SQRT(T1)
        D1 = -A1 + T2
        D2 = -A1 - T2
C       IF(J.EQ.JA) D2=D1=-2.*A1                                        OLD
        IF(J.EQ.JA) THEN                                                NEW
           D1=-2.*A1                                                    NEW
           D2=-2.*A1                                                    NEW
        ENDIF                                                           NEW
        GO TO 290
  280   D2 = -D1
  290   IF(D1.LE.0.) GO TO 300
        IF(D2.GT.0.) D1=D2
        IF(D1.GE.DLSS) GO TO 300
        JAS = J
        DLSS = D1
  300   CONTINUE
        DLS = DLSS+1.0E-10
        JA = JAS
        RETURN
   19   DO 301 J=2,4
        D1 = -1.0
        GO TO (56,161,51,56),J
   51   IF(V.EQ.0.) GO TO 301
        D1 = (CL-Y)/V
        GO TO 281
   56   IF(V.EQ.0.) GO TO 301
        D1 = (CL2-Y)/V
        GO TO 281
  161   T1 = U**2 + W**2
        IF(T1.EQ.0.) GO TO 301
        A1 = (X*U + Z*W)/T1
        B1 = (X**2 + Z**2 - CRAD2)/T1
        T1 = A1**2 - B1
        IF(T1.LT.0.) GO TO 301
```

```
      T2 = SQRT(T1)
      D1 = -A1 + T2
      D2 = -A1 - T2
C     IF (J.EQ.JA) D2=D1=-2.*A1                                        OLD
      IF (J.EQ.JA) THEN                                                NEW
          D1=-2.*A1                                                    NEW
          D2=-2.*A1                                                    NEW
      END IF                                                           NEW
      GO TO 291
  281 D2 = -D1
  291 IF(D1.LE.0.) GO TO 301
      IF(D2.GT.0.) D1=D2
      IF(D1.GE.DLSS) GO TO 301
      JAS = J
      DLSS = D1
  301 CONTINUE
      DLS = DLSS+1.0E-10
      JA = JAS
      RETURN
      END

      SUBROUTINE KLEIN(T1,T4,KRN)                                      ARG
C        SAMPLE FROM KLEIN-NISHINA USING INVERSE FIT.
C        T1=ENERGY IN, T4=ENERGY OUT, IN UNITS OF THE REST MASS
C        OF AN ELECTRON.
C
      RN=RANF(KRN)
      T2=1./T1
      T4=2.*T1+1.
      T5=1./T4
      T6=ALOG(T4)
      T3=2.*T1*(1.+T1)*T5**2+4.*T2+(1.-2.*T2*(1.+T2))*T6
      IF(T1.LE.1.16666667)GO TO 20
      T7=1.65898+T2*(.62537*T2-1.00796)
      T3=T7/T3
      IF(RN.LE.T3)GO TO 10
      T4=(T6-1.20397)/(1.-T3)
      T7=.3*EXP(T4*(T3-RN))
      GO TO 40
   10 T4=T7/(3.63333+T2*(5.44444*T2-4.66667))
      T7=.5*T7
      T2=RN/T3
      T3=2.1
      T5=1.4
      GO TO 30
   20 T4=T3/(T4+T5)
      T7=.5*T3
      T2=RN
      T5=1.-T5
      T3=3.*T5
      T5=2.*T5
   30 T7=1.+T2*(T2*(2.*T7+T4-T3+T2*(T5-T7-T4))-T7)
   40 T4=T7*T1
      RETURN
      END
```

```
      SUBROUTINE ISOS(U,V,W,KRN)                                        ARGS
C        SAMPLE A DIRECTION U,V,W ISOTROPICALLY.
C
   10 T1=2.*RANF(KRN)-1.
      T2=2.*RANF(KRN)-1.
      RSQ=T1**2+T2**2
      IF(RSQ.GT.1.0)GO TO 10
      U=2.*RSQ-1.
      T3=SQRT((1.-U**2)/RSQ)
      V=T1*T3
      W=T2*T3
      RETURN
      END

      SUBROUTINE ROTAS(C,KRN,U,V,W,UOLD,VOLD,WOLD)                      ARGS
C        ROTATE UOLD,VOLD,WOLD TO U,V,W THROUGH A POLAR
C        ANGLE WHOSE COSINE IS C, AND THROUGH AN AZIMUTHAL
C        ANGLE SAMPLED UNIFORMLY.
C
   10 T1=2.*RANF(KRN)-1.
      T2=2.*RANF(KRN)-1.
      R=T1**2+T2**2
      IF(R.GT.1.0)GO TO 10
      R=SQRT((1.-C**2)/R)
      T1=T1*R
      T2=T2*R
      IF(ABS(WOLD).GT..999999)GO TO 30
      S=SQRT(1.-WOLD**2)
      U=UOLD*C+(T1*UOLD*WOLD-T2*VOLD)/S
      V=VOLD*C+(T1*VOLD*WOLD+T2*UOLD)/S
      W=WOLD*C-T1*S
      RETURN
   30 U=T1
      V=T2
      W=WOLD*C
      RETURN
      END

      REAL FUNCTION RANF(KERN)                                          ALG CH
      KERN = MOD(1+9621*KERN,131072)                                    ALG CH
      RANF = FLOAT(KERN)/131072.                                        ALG CH
      RETURN                                                            ALG CH
      END                                                               ALG CH
```

```
 1  program_(p18,[p02])
 2  begin_
 3        clear_(d14)
 4        NPC=NPC+1
 5        IF (NPC.GE.NPPC) THEN
 6           set_(d06)
 7           clear_(d11)
 8        ENDIF
 9        suspend_
10  end_(p18)
```

APPENDIX E

Generated Parallel FORTRAN for HEP/UPX

(result of LGDF macro-expansion)

```
      C--<<<p00.f>>>
      C
            PROGRAM GAMTEB
      C
      C----- Scalar Monte Carlo Transport Code
      C
      C---------------- (HEPUNIX FORTRAN SYSTEM TABLES) ----------------
            COMMON /SYSTAB/ $LTR,$DW,$DR,LPR(40),LPS(40),LPX(40),LNL(40)
            LOGICAL $LTR,$DW(15),$DR(15)
      C
      C-------------------- (FORTRAN TRACE TABLES) ---------------------
            COMMON /TRCTAB/ LDM(15),LPM(40),PL,DL,PS,DS,DT,PT
            CHARACTER*50 PL(0:18),DL(0:15)
            CHARACTER*10 PS(0:18),DS(0:15)
            CHARACTER*4 DT(15),PT(40)
      C-----------------------------------------------------------------
            LOGICAL GO
      C
            DT(1)='d01'
               LDM(1)=01
            DT(2)='d09'
               LDM(2)=09
            DT(3)='d15'
               LDM(3)=15
            DT(4)='d07'
               LDM(4)=07
            DT(5)='d03'
               LDM(5)=03
            DT(6)='d04'
               LDM(6)=04
            DT(7)='d05'
               LDM(7)=05
            DT(8)='d06'
               LDM(8)=06
            DT(9)='d02'
               LDM(9)=02
            DT(10)='d10'
               LDM(10)=10
            DT(11)='d11'
               LDM(11)=11
            DT(12)='d12'
               LDM(12)=12
            DT(13)='d06a'
               LDM(13)=06
            DT(14)='d13'
               LDM(14)=13
            DT(15)='d14'
               LDM(15)=14
      C
            PT(1)='top'
               LPM(1)= -1
            PT(2)='p00'
               LPM(2)=00
            PT(3)='p10'
               LPM(3)=10
            PT(4)='p01'
```

```
      LPM(4)=01
    PT(5)='p12'
      LPM(5)=12
    PT(6)='p13'
      LPM(6)=13
    PT(7)='p02'
      LPM(7)=02
    PT(8)='p14'
      LPM(8)=14
    PT(9)='p16'
      LPM(9)=16
    PT(10)='p17'
      LPM(10)=17
    PT(11)='p17a'
      LPM(11)=17
    PT(12)='p17b'
      LPM(12)=17
    PT(13)='p17c'
      LPM(13)=17
    PT(14)='p17d'
      LPM(14)=17
    PT(15)='p17e'
      LPM(15)=17
    PT(16)='p17f'
      LPM(16)=17
    PT(17)='p17g'
      LPM(17)=17
    PT(18)='p17h'
      LPM(18)=17
    PT(19)='p17i'
      LPM(19)=17
    PT(20)='p17j'
      LPM(20)=17
    PT(21)='p17k'
      LPM(21)=17
    PT(22)='p17l'
      LPM(22)=17
    PT(23)='p17m'
      LPM(23)=17
    PT(24)='p17n'
      LPM(24)=17
    PT(25)='p17o'
      LPM(25)=17
    PT(26)='p17p'
      LPM(26)=17
    PT(27)='p17q'
      LPM(27)=17
    PT(28)='p17s'
      LPM(28)=17
    PT(29)='p17r'
      LPM(29)=17
    PT(30)='p17s'
      LPM(30)=17
    PT(31)='p17t'
      LPM(31)=17
    PT(32)='p17u'
```

```
      LPM(32)=17
     PT(33)='p17v'
      LPM(33)=17
     PT(34)='p17w'
      LPM(34)=17
     PT(35)='p17x'
      LPM(35)=17
     PT(36)='p17y'
      LPM(36)=17
     PT(37)='p17z'
      LPM(37)=17
     PT(38)='p17'
      LPM(38)=17
     PT(39)='p17'
      LPM(39)=17
     PT(40)='p18'
      LPM(40)=18
C
     DS(01)='user input'
      DL(01)='No. of Particles to run'
     DS(02)='NPP'
      DL(02)='No. of Particles to run'
     DS(03)='constants'
      DL(03)='problem constants'
     DS(04)='cs tables'
      DL(04)='cross section tables'
     DS(05)='src values'
      DL(05)='source values'
     DS(06)='run stats'
      DL(06)='run statistics'
     DS(07)='report'
      DL(07)='GAMTEB report file'
     DS(09)='prompts'
      DL(09)='prompt for no. of particles'
     DS(10)='NPPR'
      DL(10)='particle count for report'
     DS(11)='NPPC;NPC'
      DL(11)='particle completion control'
     DS(12)='PSC'
      DL(12)='particle start control'
     DS(13)='KERN'
      DL(13)='random seed'
     DS(14)='signal'
      DL(14)='particle history completion signal'
     DS(15)='done'
      DL(15)='run completion interlock'
C
     PS(00)='GAMTEB'
      PL(00)='Scalar Monte Carlo Transport Code'
     PS(01)='RUNPR'
      PL(01)='run particle histories and print report'
     PS(02)='RUNPH'
      PL(02)='run particle history'
     PS(10)='SETUP'
      PL(10)='set up problem constants'
     PS(12)='PARAM'
```

```
      PL(12)='init GAMTEB run parameters'
      PS(13)='PCOUNT'
      PL(13)='setup for particle counting'
      PS(14)='REPORT'
      PL(14)='format GAMTEB report'
      PS(16)='GENRAN'
      PL(16)='generate random seed for next particle'
      PS(17)='RNHIST'
      PL(17)='run history for one particle and offspring'
      PS(18)='CKCOMP'
      PL(18)='check for run completion'
C
C     OPEN(4,FILE='tf')
C
      I=1
 7720 LPR(I)=0
      LPX(I)=0
      LPS(I)=0
      LNL(I)=0
      I=I+1
      IF(I.LE.40) GO TO  7720
C
      I=1
 7730 CALL SETE($DW(I))
      CALL SETE($DR(I))
      CALL AWRITE($DW(I),.FALSE.)
      CALL AWRITE($DR(I),.FALSE.)
      CALL SETE($DW(I))
      CALL SETE($DR(I))
      I=I+1
      IF(I.LE.15) GO TO  7730
C
      GO=LAREAD($LTR)
      CALL CLOCK(ISTART)
      CALL PTRACE(2,42,0,ISTART,0.0)
      CALL AWRITE($LTR,GO)
C
C---- START LGDF EXECUTION ----
C
        CALL P00(2,1)
C
C---- AWAIT RESULTS ----
C
      CALL AWRITE($DR(3),LAREAD($DR(3)))
      GO=LAREAD($LTR)
      CALL PTRACE(2,40,0,0,0.0)
C
      CALL CLOCK(IEND)
      CALL PTRACE(2,43,0,IEND-ISTART,0.0)
      CALL PTRACE(2,50,0,0,0.0)
C
      I=2
 7797 IF(LPX(I).GT.0) CALL PTRACE(I,51,0,LPX(I),0.0)
      IF(LPS(I).GT.0) CALL PTRACE(I,52,0,LPS(I),0.0)
      IF(LNL(I).GT.0) CALL PTRACE(I,53,0,LNL(I),0.0)
      IF(LPR(I).EQ. -1) CALL PTRACE(I,54,0,0,0.0)
```

```
      I=I+1
      IF(I.LE.40) GO TO  7797
C
      CALL PTRACE(2,55,0,0,0.0)
C
      I=1
 7798 IF(FULL($DR(I))) THEN
         IF(VALUE($DR(I))) CALL PTRACE(2,56,I,0,0.0)
         IF(.NOT.VALUE($DR(I))) CALL PTRACE(2,57,I,0,0.0)
      ENDIF
      I=I+1
      IF(I.LE.15) GO TO  7798
C
      STOP
      END
C
      SUBROUTINE PTRACE(IPN,IFC,IDN,IVAL,RVAL)
C
C---------------- (HEPUNIX FORTRAN SYSTEM TABLES) ----------------
      COMMON /SYSTAB/ $LTR,$DW,$DR,LPR(40),LPS(40),LPX(40),LNL(40)
      LOGICAL $LTR,$DW(15),$DR(15)
C
C-------------------- (FORTRAN TRACE TABLES) --------------------
      COMMON /TRCTAB/ LDM(15),LPM(40),PL,DL,PS,DS,DT,PT
      CHARACTER*50 PL(0:18),DL(0:15)
      CHARACTER*10 PS(0:18),DS(0:15)
      CHARACTER*4 DT(15),PT(40)
C-----------------------------------------------------------------
C
      IF(IFC.EQ.40) WRITE(4,40) PT(IPN)
 40   FORMAT(' ',/,' ',A4,': ','*** LGDE -- NORMAL TERMINATION ***')
      IF(IFC.EQ.41) WRITE(4,41) PT(IPN),DT(IDN),DS(LDM(IDN))
 41   FORMAT(' ',A4,': ',' UNLATCHES ',A4,': ',A10)
      IF(IFC.EQ.42) WRITE(4,42) PT(IPN),IVAL
 42   FORMAT(' ',A4,': ','HEP CLOCK START TIME =',I20,
     +   ' (100 NS CYCLES)')
      IF(IFC.EQ.43) WRITE(4,43) PT(IPN),IVAL
 43   FORMAT(' ',/,' ',A4,': ','HEP ELAPSED TIME =',I12,
     +   ' (100 NS CYCLES)')
      IF(IFC.EQ.44) WRITE(4,44) PT(IPN),RVAL
 44   FORMAT(' ',/,' ',A4,': ','VAX ELAPSED TIME =',F6.2,' SECS.')
      IF(IFC.EQ.45) WRITE(4,45) PT(IPN),RVAL
 45   FORMAT(' ',A4,': ',' USER TIME =',F6.2,' SECS.')
      IF(IFC.EQ.46) WRITE(4,46) PT(IPN),RVAL
 46   FORMAT(' ',A4,': ',' SYSTEM TIME =',F6.2,' SECS.')
      IF(IFC.EQ.47) WRITE(4,47) PT(IPN),PS(LPM(IPN)),PL(LPM(IPN)),
     +   PT(IPN),LPS(IPN),IVAL
 47   FORMAT(' ',/,' ',A4,': ',A10,' --- ',A50,/,' ',
     +   A4,':     $0',I1,'          (EXEC START ',I8,')')
      IF(IFC.EQ.48) WRITE(4,48) PT(IPN),PS(LPM(IPN)),PT(IPN),IVAL
 48   FORMAT(' ',A4,': ',A10,' *** TERMINATED !!! ',/,' ',
     +   A4,':                (EXEC START ',I10,')')
      IF(IFC.EQ.49) WRITE(4,49) PT(IPN)
 49   FORMAT(' ',A4,': ','*** LGDF DONE',
     +   ' -- BUT NOT ALL EXPECTED D_S ARE SET !!! ')
      IF(IFC.EQ.50) WRITE(4,50) PT(IPN)
```

```
281    50   FORMAT(' ',/,' ',A4,': ',
282        +   '--------------RUN STATISTICS--------------')
283         IF(IFC.EQ.51) WRITE(4,51) PT(IPN),IVAL
284    51   FORMAT(' ',/,' ',A4,': ','EXEC COUNT=',I10)
285         IF(IFC.EQ.52) WRITE(4,52) PT(IPN),IVAL
286    52   FORMAT(' ',A4,': ','    PSTATE= s0',I1)
287         IF(IFC.EQ.53) WRITE(4,53) PT(IPN),IVAL
288    53   FORMAT(' ',A4,': ','NO. LATCHES=',I2)
289         IF(IFC.EQ.54) WRITE(4,54) PT(IPN)
290    54   FORMAT(' ',A4,': ','*** TERMINATED! ***')
291         IF(IFC.EQ.55) WRITE(4,55) PT(IPN)
292    55   FORMAT(' ',/,' ',A4,': ','--- SET D S ---',/)
293         IF(IFC.EQ.56) WRITE(4,56) PT(IPN),DT(IDN),DS(LDM(IDN))
294    56   FORMAT(' ',A4,': ',A4,': ',A10,' /EOF')
295         IF(IFC.EQ.57) WRITE(4,57) PT(IPN),DT(IDN),DS(LDM(IDN))
296    57   FORMAT(' ',A4,': ',A4,': ',A10)
297         IF(IFC.EQ.58) WRITE(4,58) PT(IPN),IVAL
298    58   FORMAT(' ',A4,': ','REITERATES ... (STATE = s0',I1,')')
299         IF(IFC.EQ.59) WRITE(4,59) PT(IPN),DT(IDN),DS(LDM(IDN))
300    59   FORMAT(' ',A4,': ','          ASETS ',A4,': ',A10)
301         IF(IFC.EQ.60) WRITE(4,60) PT(IPN),DT(IDN),DS(LDM(IDN))
302    60   FORMAT(' ',A4,': ','           SETS ',A4,': ',A10)
303         IF(IFC.EQ.61) WRITE(4,61) PT(IPN)
304    61   FORMAT(' ',A4,': ',16X,'**AUTOLATCHED**')
305         IF(IFC.EQ.62) WRITE(4,62) PT(IPN),DT(IDN),DS(LDM(IDN))
306    62   FORMAT(' ',A4,': ','         CLEARS ',A4,': ',A10)
307         IF(IFC.EQ.63) WRITE(4,63) PT(IPN),DT(IDN),DS(LDM(IDN))
308    63   FORMAT(' ',A4,': ','        ACLEARS ',A4,': ',A10)
309         IF(IFC.EQ.64) WRITE(4,64) PT(IPN),DT(IDN),DS(LDM(IDN))
310    64   FORMAT(' ',A4,': ','         AREADS ',A4,': ',A10)
311         IF(IFC.EQ.65) WRITE(4,65) PT(IPN),DT(IDN),DS(LDM(IDN))
312    65   FORMAT(' ',A4,': ','        AWRITES ',A4,': ',A10)
313         IF(IFC.EQ.66) WRITE(4,66) PT(IPN),DT(IDN),DS(LDM(IDN))
314    66   FORMAT(' ',A4,': ','   SETS EOF ON ',A4,': ',A10)
315         IF(IFC.EQ.67) WRITE(4,67) PT(IPN),DT(IDN),DS(LDM(IDN))
316    67   FORMAT(' ',A4,': ',' CLEARS EOF ON ',A4,': ',A10)
317         IF(IFC.EQ.68) WRITE(4,68) PT(IPN),IVAL
318    68   FORMAT(' ',A4,': ','   NEXT STATE  s0',I1)
319         IF(IFC.EQ.69) WRITE(4,69) PT(IPN),PS(LPM(IPN)),PL(LPM(IPN)),
320        +   PT(IPN),IVAL
321    69   FORMAT(' ',/,' ',A4,': ',A10,' === ',A50,/,' ',
322        +   A4,':                    (EXEC START ',I8,')')
323         IF(IFC.EQ.70) WRITE(4,70) PT(IPN),IVAL
324    70   FORMAT(' ',A4,': ','       SUSPENDS (EXEC START',I8,')')
325         IF(IFC.EQ.71) WRITE(4,71) PT(IPN),IDN,IVAL
326    71   FORMAT(' ',A4,': ',
327        +   '       ERROR IN AUTO-UNLATCHING !!! LNL(IPN) =',
328        +   I2,' (EXEC START',I8,')')
329         RETURN
330         END
331   C
332   C
333   C---- HEPUNIX FORTRAN SCHEDULERS ----
334   C
335   C
336   C
```

```
      SUBROUTINE P00(IPN,IPCTXT)
C
C
C---------------- (HEPUNIX FORTRAN SYSTEM TABLES) ----------------
      COMMON /SYSTAB/ $LTR,$DW,$DR,LPR(40),LPS(40),LPX(40),LNL(40)
      LOGICAL $LTR,$DW(15),$DR(15)
C
      EXTERNAL P10
 7798 LPX(IPN)=LPX(IPN)+1
           CALL CREATE(P10,3,IPN)
        CALL P01(4,IPN)
      RETURN
      END
C
      SUBROUTINE P01(IPN,IPCTXT)
C
C
C---------------- (HEPUNIX FORTRAN SYSTEM TABLES) ----------------
      COMMON /SYSTAB/ $LTR,$DW,$DR,LPR(40),LPS(40),LPX(40),LNL(40)
      LOGICAL $LTR,$DW(15),$DR(15)
C
      EXTERNAL P12
      EXTERNAL P13
      EXTERNAL P14
 7798 LPX(IPN)=LPX(IPN)+1
           CALL CREATE(P12,5,IPN)
      CALL CREATE(P13,6,IPN)
        CALL P02(7,IPN)
      CALL CREATE(P14,8,IPN)
      RETURN
      END
C
      SUBROUTINE P02(IPN,IPCTXT)
C
C
C---------------- (HEPUNIX FORTRAN SYSTEM TABLES) ----------------
      COMMON /SYSTAB/ $LTR,$DW,$DR,LPR(40),LPS(40),LPX(40),LNL(40)
      LOGICAL $LTR,$DW(15),$DR(15)
C
      EXTERNAL P16
      EXTERNAL P17
      EXTERNAL P18
 7798 LPX(IPN)=LPX(IPN)+1
           CALL CREATE(P16,9,IPN)
      CALL CREATE(P17,10,IPN)
      CALL CREATE(P17,11,IPN)
      CALL CREATE(P17,12,IPN)
      CALL CREATE(P17,13,IPN)
      CALL CREATE(P17,14,IPN)
      CALL CREATE(P17,15,IPN)
      CALL CREATE(P17,16,IPN)
      CALL CREATE(P17,17,IPN)
      CALL CREATE(P17,18,IPN)
      CALL CREATE(P17,19,IPN)
      CALL CREATE(P17,20,IPN)
      CALL CREATE(P17,21,IPN)
```

```
      CALL CREATE(P17,22,IPN)
      CALL CREATE(P17,23,IPN)
      CALL CREATE(P17,24,IPN)
      CALL CREATE(P17,25,IPN)
      CALL CREATE(P17,26,IPN)
      CALL CREATE(P17,27,IPN)
      CALL CREATE(P17,28,IPN)
      CALL CREATE(P17,29,IPN)
      CALL CREATE(P17,30,IPN)
      CALL CREATE(P17,31,IPN)
      CALL CREATE(P17,32,IPN)
      CALL CREATE(P17,33,IPN)
      CALL CREATE(P17,34,IPN)
      CALL CREATE(P17,35,IPN)
      CALL CREATE(P17,36,IPN)
      CALL CREATE(P17,37,IPN)
      CALL CREATE(P17,38,IPN)
      CALL CREATE(P17,39,IPN)
      CALL CREATE(P18,40,IPN)
      RETURN
      END
C==>END: p00.f
```

```
      C--<<<p10.f>>>
      C
      C---- SETUP -- set up problem constants
      C
            SUBROUTINE P10(IPN,IPCTXT)
      C
      C
      C--------------- (HEPUNIX FORTRAN SYSTEM TABLES) ---------------
            COMMON /SYSTAB/ $LTR,$DW,$DR,LPR(40),LPS(40),LPX(40),LNL(40)
            LOGICAL $LTR,$DW(15),$DR(15)
      C
      C-> d03: constants  - problem constants
            COMMON /D03/ E,RHO,CL,CL2,CRAD,CRAD2,WCP1,WCP2,EC,FIM
            REAL E(35),RHO,CL,CL2,CRAD,CRAD2,WCP1,WCP2,EC,FIM(2)
      C-> d04: cs tables  - cross section tables
            COMMON /D04/ EL,XC,XPP,XPE
            REAL EL(35),XC(35),XPP(35),XPE(35)
      C-> d05: src values - source values
            COMMON /D05/ GIERG,GIWT,GIU,GIV,GIW,GIX,GIY,GIZ,IGIA
            REAL GIERG,GIWT,GIU,GIV,GIW,GIX,GIY,GIZ
            INTEGER IGIA
            LOGICAL GO,DFPROG
      C
            IF(LPR(IPN).EQ. -1) RETURN
       7799 DFPROG=.FALSE.
            CALL AWRITE($DW(5),GO)
            CALL AWRITE($DW(6),LAREAD($DW(5)))
            CALL AWRITE($DW(7),LAREAD($DW(6)))
            GO=LAREAD($DW(7))
        7798 LPX(IPN)=LPX(IPN)+1
            LPR(IPN)=1
            LPR(IPCTXT)=LPR(IPCTXT)+1
      C
            CL=20.0
            CL2=CL+10.
            CRAD=1.0
            CRAD2=CRAD**2
            WCP1=.5
            WCP2=.25
            EC = .001
            FIM(1)=1.0                                                  STYLE
            FIM(2)=2.0
                  CALL AWRITE($DW(5),.FALSE.)
               DFPROG=.TRUE.
                  CALL AWRITE($DR(5),.FALSE.)
               DFPROG=.TRUE.
      C
      C        CONVERT CROSS-SECTION UNITS TO BE PER CM.
      C
            DO 1 I=1,35
            XC(I)=ALOG( XC(I)*RHO )
            IF(XPP(I).EQ.0.) XPP(I)=1.0E-37                             NEW
            IF(XPE(I).EQ.0.) XPE(I)=1.0E-37                             NEW
            XPP(I)=ALOG( XPP(I)*RHO )
            XPE(I)=ALOG( XPE(I)*RHO )
```

```
      EL(I)=ALOG(E(I))
    1 CONTINUE
         CALL AWRITE($DW(6),.FALSE.)
         DFPROG=.TRUE.
         CALL AWRITE($DR(6),.FALSE.)
         DFPROG=.TRUE.
C
C        SET SOURCE VALUES
C
      GIERG = 6.0
      GIWT = 1.0
      GIU = 0.0
      GIV = 1.0
      GIW = 0.0
      GIX = 0.0
      GIY = .000001
      GIZ = 0.0
      IGIA=1
         CALL AWRITE($DW(7),.FALSE.)
         DFPROG=.TRUE.
         CALL AWRITE($DR(7),.FALSE.)
         DFPROG=.TRUE.
         GO TO 7797
 7797 IF(.NOT.DFPROG) THEN
      GO=LAREAD($LTR)
      CALL PTRACE(IPN,48,0,LPX(IPN),0.0)
      CALL AWRITE($LTR,VALUE($LTR))
         LPR(IPN)= -1
         RETURN
      ENDIF
      GO TO 7799
      END
C==>END:p10.f
      BLOCK DATA
C-> d03: constants - problem constants
      COMMON /D03/ E,RHO,CL,CL2,CRAD,CRAD2,WCP1,WCP2,EC,FIM
      REAL E(35),RHO,CL,CL2,CRAD,CRAD2,WCP1,WCP2,EC,FIM(2)

C-> d04: cs tables - cross section tables
      COMMON /D04/ EL,XC,XPP,XPE
      REAL EL(35),XC(35),XPP(35),XPE(35)

      DATA RHO/2.22/
      DATA (E(I),I=1,35)/ .001, .0015, .002, .003, .004, .005,
     1    .006, .008, .01, .015, .02, .03, .04, .05, .06, .08,
     2    .1, .15, .2, .3, .4, .5, .6, .8, 1., 1.5, 2., 3., 4., 5.,
     3    6., 8., 10., 15., 20./
      DATA (XC(I),I=1,35)/.0150,.0296,.0451,.0717,.0913
     1,.105,.115,.128,.137,.152
     2,.160,.165,.165,.163,.160
     3,.153,.146,.133,.122,.106,.0953,.0867,.0802,.0707,.0637
     4,.0516,.044,.0346,.0289,.0250,.0221,.0181,.0154,.0114,.00913/
      DATA (XPP(I),I=1,35)/ 0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,
     1    0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,.0000792,
     2    .000316,.000923,.00153,.00208,.00256,.00343,
     3    .00414, .00547, .00652/
```

```
      DATA (XPE(I),I=1,35)/ 2010., 632., 280., 87.7, 37.3, 18.9,
     1     10.4, 4.01, 1.91, .489, .192, .0491, .0186, .00887,
     2     .00481, .00179, .000862, .000234, .0000918,
     4  0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0./
      END
```

```
      1  C--<<<p12.f>>>
      2  C
      3  C---- PARAM -- init GAMTEB run parameters
      4  C
      5        SUBROUTINE P12(IPN,IPCTXT)
      6  C
      7  C
      8  C--------------- (HEPUNIX FORTRAN SYSTEM TABLES) ---------------
      9        COMMON /SYSTAB/ $LTR,$DW,$DR,LPR(40),LPS(40),LPX(40),LNL(40)
     10        LOGICAL $LTR,$DW(15),$DR(15)
     11  C
     12  C-> d01: user input - No. of Particles to run
     13        COMMON /D01/ ID01
     14        INTEGER ID01
     15
     16  C-> d09: prompts    - prompt for no. of particles
     17        COMMON /D09/ CL,CL2,CRAD,CRAD2,WCP1,WCP2,EC
     18        REAL CL,CL2,CRAD,CRAD2,WCP1,WCP2,EC
     19  C-> d15: done       - run completion interlock
     20        COMMON /D15/ ID015
     21        INTEGER ID015
     22  C-> d06: run stats  - run statistics
     23        COMMON /D06/ IGNCOL,IGNCO,GWCO,GWCP,GWRL,IGNR,GWRG,IGCUTF,
     24       +    GABSOR,IGNS,
     25       +    TRANS,ESCAPE,BSCAT,TRANS2,ESCAP2,BSCAT2,
     26       +    BTOT2,TTOT2,ETOT2,ETOT,BTOT,TTOT
     27        INTEGER IGNCOL,IGNCO,IGNR,IGCUTF,IGNS
     28        REAL GWCO,GWCP,GWRL,GWRG,GABSOR,
     29       +    TRANS(35),ESCAPE(35),BSCAT(35),
     30       +    TRANS2(35),ESCAP2(35),BSCAT2(35),
     31       +    BTOT2,TTOT2,ETOT2,ETOT,BTOT,TTOT
     32  C-> d02: NPP        - No. of Particles to run
     33        COMMON /D02/ NPP
     34        INTEGER NPP
     35        LOGICAL GO,DFPROG
     36
     37        IF(LPR(IPN).EQ.-1) RETURN
     38   7799 DFPROG=.FALSE.
     39        CALL AWRITE($DW(3),GO)
     40        CALL AWRITE($DW(8),LAREAD($DW(3)))
     41        CALL AWRITE($DW(9),LAREAD($DW(8)))
     42        GO=LAREAD($DW(9))
     43   7798 LPX(IPN)=LPX(IPN)+1
     44        LPR(IPN)=1
     45        LPR(IPCTXT)=LPR(IPCTXT)+1
     46
     47  C
     48  C         INITIALIZE PROBLEM INPUT
     49  C
     50        GO=LAREAD($LTR)
     51        WRITE(6,*) 'p12: ENTER NO. OF PARTICLES TO RUN'                  UPGRADE
     52  C     NPP=500000                                                       OLD
     53        READ(5,*) NPP                                                    UPGRADE
     54        CALL AWRITE($LTR,GO)
     55        IF (NPP.EQ.0) THEN
     56                CALL AWRITE($DW(3),.FALSE.)
```

```
      DFPROG=.TRUE.
         CALL AWRITE($DR(3),.FALSE.)
      DFPROG=.TRUE.
            GO TO  7797
   ELSE
            CALL AWRITE($DW(9),.FALSE.)
      DFPROG=.TRUE.
         CALL AWRITE($DR(9),.FALSE.)
      DFPROG=.TRUE.
   ENDIF
C
   IGNCOL=0
   IGNCO=0
   GWCO=0
   GWCP=0
   GWRL=0.
   GWRG=0.
   GABSOR=0.
   IGNR=0
   IGCUTF= 0
   IGNS=0                                                    BUGFIX
   DO 5 I = 1,35
      TRANS2(I)= 0.0
      BSCAT2(I)= 0.0
      ESCAP2(I)= 0.0                                         RENAME
      TRANS(I) = 0.0
      BSCAT(I) = 0.0
      ESCAPE(I) = 0.0
 5 CONTINUE                                                  STYLE
   BTOT=0.0
   BTOT2=0.0
   TTOT=0.0
   TTOT2=0.0
   ETOT=0.0
   ETOT2=0.0
         CALL AWRITE($DW(8),.FALSE.)
      DFPROG=.TRUE.
         CALL AWRITE($DR(8),.FALSE.)
      DFPROG=.TRUE.
         GO TO  7797
7797 IF(.NOT.DFPROG) THEN
   GO=LAREAD($LTR)
   CALL PTRACE(IPN,48,0,LPX(IPN),0.0)
   CALL AWRITE($LTR,VALUE($LTR))
      LPR(IPN)= -1
      RETURN
   ENDIF
   GO TO  7799
   END
C==>END:p12.f
```

```
      1  C--<<<p13.f>>>
      2  C
      3  C---- PCOUNT -- setup for particle counting
      4  C
      5        SUBROUTINE P13(IPN,IPCTXT)
      6  C
      7  C
      8  C--------------- (HEPUNIX FORTRAN SYSTEM TABLES) --------------
      9        COMMON /SYSTAB/ $LTR,$DW,$DR,LPR(40),LPS(40),LPX(40),LNL(40)
     10        LOGICAL $LTR,$DW(15),$DR(15)
     11  C
     12  C-> d02: NPP          - No. of Particles to run
     13        COMMON /D02/ NPP
     14        INTEGER NPP
     15  C-> d10: NPPR         - particle count for report
     16        COMMON /D10/ NPPR
     17        INTEGER NPPR
     18  C-> d11: NPPC,NPC     - particle completion control
     19        COMMON /D11/ NPC,NPPC
     20        INTEGER NPC,NPPC
     21  C-> d12: PSC          - particle start control
     22        COMMON /D12/ NPPS,NPS,KRN2
     23        INTEGER NPPS,NPS,KRN2
     24        LOGICAL GO,DFPROG
     25
     26        IF(LPR(IPN).EQ. -1) RETURN
     27   7799 DFPROG=.FALSE.
     28        CALL AWRITE($DR(9),LAREAD($DR(9)))
     29        CALL AWRITE($DW(10),GO)
     30        CALL AWRITE($DW(11),LAREAD($DW(10)))
     31        CALL AWRITE($DW(12),LAREAD($DW(11)))
     32        GO=LAREAD($DW(12))
     33   7798 LPX(IPN)=LPX(IPN)+1
     34        LPR(IPN)=1
     35        LPR(IPCTXT)=LPR(IPCTXT)+1
     36
     37  C
     38  C
     39  C     KRN = 123454321                                          OLD
     40  C -- CHANGE TO KRN2 DUE TO ADDITION OF SECOND R.N.GENERATOR    ALG CH
     41  C -- SET UP PARTICLE START CONTROL                             NEW
     42        KRN2 = 123                                               ALG CH
     43        NPPS=NPP                                                 NEW
     44        NPS=0
     45             CALL AWRITE($DW(12),.FALSE.)
     46           DFPROG=.TRUE.
     47             CALL AWRITE($DR(12),.FALSE.)
     48           DFPROG=.TRUE.
     49  C -- SET UP PARTICLE COMPLETION CONTROL
     50        NPC=0                                                    NEW
     51        NPPC=NPP                                                 NEW
     52             CALL AWRITE($DW(11),.FALSE.)
     53           DFPROG=.TRUE.
     54             CALL AWRITE($DR(11),.FALSE.)
     55           DFPROG=.TRUE.
     56  C -- MAKE COPY OF NPP FOR REPORT PROCESS
```

```
      NPPR=NPP
         CALL AWRITE($DW(10),.FALSE.)
      DFPROG=.TRUE.
         CALL AWRITE($DR(10),.FALSE.)
      DFPROG=.TRUE.
         GO=LAREAD($DR(9))
         GO=LAREAD($DW(9))
      DFPROG=.TRUE.
         GO TO  7797
 7797 IF(.NOT.DFPROG) THEN
      GO=LAREAD($LTR)
      CALL PTRACE(IPN,48,0,LPX(IPN),0.0)
      CALL AWRITE($LTR,VALUE($LTR))
         LPR(IPN)= -1
         RETURN
      ENDIF
      GO TO  7799
      END
C==>END:p13.f
```

```
      C--<<<p14.f>>>
      C
      C---- REPORT -- format GAMTEB report
      C
            SUBROUTINE P14(IPN,IPCTXT)
      C
      C
      C--------------- (HEPUNIX FORTRAN SYSTEM TABLES) ---------------
            COMMON /SYSTAB/ $LTR,$DW,$DR,LPR(40),LPS(40),LPX(40),LNL(40)
            LOGICAL $LTR,$DW(15),$DR(15)
      C
      C-> d06: run stats  - run statistics
            COMMON /D06/ IGNCOL,IGNCO,GWCO,GWCP,GWRL,IGNR,GWRG,IGCUTF,
           +   GABSOR,IGNS,
           +   TRANS,ESCAPE,BSCAT,TRANS2,ESCAP2,BSCAT2,
           +   BTOT2,TTOT2,ETOT2,ETOT,BTOT,TTOT
            INTEGER IGNCOL,IGNCO,IGNR,IGCUTF,IGNS
            REAL GWCO,GWCP,GWRL,GWRG,GABSOR,
           +   TRANS(35),ESCAPE(35),BSCAT(35),
           +   TRANS2(35),ESCAP2(35),BSCAT2(35),
           +   BTOT2,TTOT2,ETOT2,ETOT,BTOT,TTOT
      C-> d10: NPPR       - particle count for report
            COMMON /D10/ NPPR
            INTEGER NPPR
      C-> d03: constants  - problem constants
            COMMON /D03/ E,RHO,CL,CL2,CRAD,CRAD2,WCP1,WCP2,EC,FIM
            REAL E(35),RHO,CL,CL2,CRAD,CRAD2,WCP1,WCP2,EC,FIM(2)
      C-> d07: report     - GAMTEB report file
            COMMON /D07/ ID07
            INTEGER ID07
            LOGICAL GO,DFPROG
      C
            REAL RTRANS(35),RBSCAT(35),RESCAP(35)                    RENAME
            IF(LPR(IPN).EQ. -1) RETURN
       7799 DFPROG=.FALSE.
            CALL AWRITE($DR(13),LAREAD($DR(13)))
            CALL AWRITE($DR(10),LAREAD($DR(10)))
            CALL AWRITE($DR(5),LAREAD($DR(5)))
       7798 LPX(IPN)=LPX(IPN)+1
            LPR(IPN)=1
            LPR(IPCTXT)=LPR(IPCTXT)+1
      C
      C
      C         PRINT OUTPUT
      C
      C -- NPP AND NPS ARE USED INTERCHANGEABLY IN THE ORIGINAL CODE     NEW
      C 140 NPS = NPS - 1                                              OLD
        140 NPS=NPPR                                                   NEW
      C -- FIX UP NAME ALIAS FOR NPP                                   NEW
            NPP=NPPR                                                   NEW
      C -- GET EXCLUSIVE ACCESS TO I/O FOR PRINT FILE                  HEP
            GO=LAREAD($LTR)
            WRITE(4,7634) IGNCOL
       7634 FORMAT(5H NCOL,I10)
            WRITE(4,1401)
```

```
      1401 FORMAT(7HSCALERT,/)
           WRITE(4,150) NPS
       150 FORMAT(6HNPS = ,I6)
           WRITE(4,200)
       200 FORMAT(///,8X,1HE,13X,5HBSCAT,9X,9HREL ERROR)
           DO 220 I=1,35
           RNPS= NPS
           TRANS(I) = TRANS(I)/RNPS
           BSCAT(I) = BSCAT(I)/RNPS
           ESCAPE(I) = ESCAPE(I)/RNPS
           TRANS2(I)=TRANS2(I)/RNPS
           BSCAT2(I)=BSCAT2(I)/RNPS
           ESCAP2(I)=ESCAP2(I)/RNPS                                    RENAME
           IF(TRANS(I).NE.0.0)GO TO 203
           RTRANS(I)= 0.0
           GO TO 204
       203 RTRANS(I)= SQRT((TRANS2(I)-TRANS(I)**2)/ RNPS)
           RTRANS(I)= RTRANS(I)/TRANS(I)
       204 IF(BSCAT(I).NE.0.0)GO TO 205
           RBSCAT(I)= 0.0
           GO TO 206
       205 RBSCAT(I)= SQRT((BSCAT2(I)-BSCAT(I)**2)/ RNPS)
           RBSCAT(I)= RBSCAT(I)/ BSCAT(I)
       206 IF(ESCAPE(I).NE.0.0)GO TO 207
           RESCAP(I)= 0.0                                              RENAME
           GO TO 209
       207 RESCAP(I)= SQRT((ESCAP2(I)-ESCAPE(I)**2)/ RNPS)             RENAME
           RESCAP(I)= RESCAP(I)/ ESCAPE(I)                             RENAME
       209 WRITE(4,210) E(I),BSCAT(I),RBSCAT(I)
       220 CONTINUE
           TTOT = TTOT/RNPS
           TTOT2 = TTOT2/RNPS
           BTOT = BTOT/RNPS
           BTOT2 = BTOT2/RNPS
           ETOT = ETOT/RNPS
           ETOT2 = ETOT2/RNPS
           IF(TTOT.NE.0.0) GO TO 2000
           RTTOT = 0.0
           GO TO 2001
      2000 RTTOT = SQRT((TTOT2 - TTOT**2)/RNPS)
           RTTOT = RTTOT/TTOT
      2001 IF(BTOT.NE.0.0) GO TO 2002
           RBTOT = 0.0
           GO TO 2003
      2002 RBTOT = SQRT((BTOT2 - BTOT**2)/RNPS)
           RBTOT = RBTOT/BTOT
      2003 IF(ETOT.NE.0.0) GO TO 2004
           RETOT = 0.0
           GO TO 2005
      2004 RETOT = SQRT((ETOT2 - ETOT**2)/RNPS)
           RETOT = RETOT/ETOT
      2005 CONTINUE
           WRITE(4,2020) BTOT, RBTOT
      2020 FORMAT(/,6X,5HTOTAL,9X,1PE10.3,5X,0PF7.4)
           WRITE(4,201)
       201 FORMAT(///,8X,1HE,13X,6HESCAPE,8X,9HREL ERROR)
```

```
      DO 225 I=1,35
      WRITE(4,210) E(I),ESCAPE(I),RESCAP(I)                          RENAME
  225 CONTINUE
      WRITE(4,2020) ETOT, RETOT
      WRITE(4,202)
  202 FORMAT(///,8X,1HE,13X,5HTRANS,9X,9HREL ERROR)
      DO 230 I=1,35
      WRITE(4,210) E(I),TRANS(I),RTRANS(I)
  210 FORMAT(5X,1PE10.3,5X,1PE10.3,5X,0PF6.3)
  230 CONTINUE
      WRITE(4,2020) TTOT, RTTOT
      GABSOR = GABSOR/NPS
      GWRG=GWRG/NPP
      GWRL=GWRL/NPP
      GWCP=GWCP/NPP
      GWCO=GWCO/NPP
      WRITE(4,221) GABSOR, IGCUTF
  221 FORMAT(////,9HABSORB = ,1PE10.3,5X,9HCUTOFF = ,I5)
      WRITE(4,3728) IGNS,IGNR
 3728 FORMAT(28H TRACKS CREATED BY SPLITTING,I8,
     1 24H TRACKS LOST TO ROULETTE,I8)
      WRITE(4,3729) GWRG,GWRL
 3729 FORMAT(27H WEIGHT CREATED BY ROULETTE,1PE11.4,
     1 24H WEIGHT LOST TO ROULETTE,1PE11.4)
      WRITE(4,3730) IGNCO
 3730 FORMAT(29H TRACKS LOST TO WEIGHT CUTOFF,I8)
      WRITE(4,3731) GWCP,GWCO
 3731 FORMAT(32H WEIGHT CREATED BY WEIGHT CUTOFF,1PE11.4,
     1 29H WEIGHT LOST TO WEIGHT CUTOFF,1PE11.4)
      WRITE(4,2021) TEND
 2021 FORMAT(///, 13HTOTAL TIME = ,1PE10.3, 8H SECONDS)
C -- RETURN ACCESS TO I/O
      CALL AWRITE($LTR,GO)
           GO=LAREAD($DR(13))
           GO=LAREAD($DW(13))
         DFPROG=.TRUE.
           GO=LAREAD($DR(10))
           GO=LAREAD($DW(10))
         DFPROG=.TRUE.
           GO TO  7797
 7797 IF(.NOT.DFPROG) THEN
      GO=LAREAD($LTR)
      CALL PTRACE(IPN,48,0,LPX(IPN),0,0)
      CALL AWRITE($LTR,VALUE($LTR))
        LPR(IPN)= -1
        RETURN
      ENDIF
      GO TO  7799
      END
C==>END: p14.f
```

```
C--<<<p16.f>>>
C
C---- GENRAN -- generate random seed for next particle
C
      SUBROUTINE P16(IPN,IPCTXT)
C
C
C--------------- (HEPUNIX FORTRAN SYSTEM TABLES) ---------------
      COMMON /SYSTAB/ $LTR,$DW,$DR,LPR(40),LPS(40),LPX(40),LNL(40)
      LOGICAL $LTR,$DW(15),$DR(15)
C
C-> d12: PSC         - particle start control
      COMMON /D12/ NPPS,NPS,KRN2
      INTEGER NPPS,NPS,KRN2
C-> d13: KERN        - random seed
      COMMON /D13/ KERN
      INTEGER KERN
      LOGICAL GO,DFPROG
C
      IF(LPR(IPN).EQ. -1) RETURN
 7799 DFPROG=.FALSE.
      CALL AWRITE($DR(12),LAREAD($DR(12)))
      CALL AWRITE($DW(14),GO)
      GO=LAREAD($DW(14))
 7798 LPX(IPN)=LPX(IPN)+1
      LPR(IPN)=1
      LPR(IPCTXT)=LPR(IPCTXT)+1

C
C   START A HISTORY
   10 NPS = NPS + 1
      IF(NPS.GT.NPPS) THEN
C        GO TO 140                                                  OLD
               GO=LAREAD($DR(12))
            GO=LAREAD($DW(12))
         DFPROG=.TRUE.
      ELSE
C        GENERATE NEW RANDOM SEED                                   ALG CH
         XJUNK = RANDO(KRN2)                                        ALG CH
         KERN = KRN2                                                ALG CH
               CALL AWRITE($DW(14),.FALSE.)
         DFPROG=.TRUE.
            CALL AWRITE($DR(14),.FALSE.)
         DFPROG=.TRUE.
      ENDIF
            GO TO  7797
 7797 IF(.NOT.DFPROG) THEN
      GO=LAREAD($LTR)
      CALL PTRACE(IPN,48,0,LPX(IPN),0.0)
      CALL AWRITE($LTR,VALUE($LTR))
         LPR(IPN)= -1
         RETURN
      ENDIF
      GO TO  7799
      END
C==>END: p16.f
```

```
      REAL FUNCTION RANDO(KERN)                                          ALG CH
      KERN = MOD(1+7421*KERN,131072)                                     ALG CH
    4 FORMAT(1X,F12.8)                                                   ALG CH
      RANDO = FLOAT(KERN)/131072.                                        ALG CH
      RETURN
      END
```

```
      1  C--<<<p17.f>>>
      2  C
      3  C---- RNHIST -- run history for one particle and offspring
      4  C
      5        SUBROUTINE P17(IPN,IPCTXT)
      6  C
      7  C
      8  C--------------- (HEPUNIX FORTRAN SYSTEM TABLES) ---------------
      9        COMMON /SYSTAB/ $LTR,$DW,$DR,LPR(40),LPS(40),LPX(40),LNL(40)
     10        LOGICAL $LTR,$DW(15),$DR(15)
     11  C
     12  C-> d06: run stats  - run statistics
     13        COMMON /D06/ IGNCOL,IGNCO,GWCO,GWCP,GWRL,IGNR,GWRG,IGCUTF,
     14       +   GABSOR,IGNS,
     15       +   TRANS,ESCAPE,BSCAT,TRANS2,ESCAP2,BSCAT2,
     16       +   BTOT2,TTOT2,ETOT2,ETOT,BTOT,TTOT
     17        INTEGER IGNCOL,IGNCO,IGNR,IGCUTF,IGNS
     18        REAL GWCO,GWCP,GWRL,GWRG,GABSOR,
     19       +   TRANS(35),ESCAPE(35),BSCAT(35),
     20       +   TRANS2(35),ESCAP2(35),BSCAT2(35),
     21       +   BTOT2,TTOT2,ETOT2,ETOT,BTOT,TTOT
     22  C-> d13: KERN       - random seed
     23        COMMON /D13/ KERN
     24        INTEGER KERN
     25  C-> d03: constants  - problem constants
     26        COMMON /D03/ E,RHO,CL,CL2,CRAD,CRAD2,WCP1,WCP2,EC,FIM
     27        REAL E(35),RHO,CL,CL2,CRAD,CRAD2,WCP1,WCP2,EC,FIM(2)
     28  C-> d04: cs tables  - cross section tables
     29        COMMON /D04/ EL,XC,XPP,XPE
     30        REAL EL(35),XC(35),XPP(35),XPE(35)
     31  C-> d05: src values - source values
     32        COMMON /D05/ GIERG,GIWT,GIU,GIV,GIW,GIX,GIY,GIZ,IGIA
     33        REAL GIERG,GIWT,GIU,GIV,GIW,GIX,GIY,GIZ
     34        INTEGER IGIA
     35  C-> d14: signal     - particle history completion signal
     36        COMMON /D14/ ID14
     37        INTEGER ID14
     38        LOGICAL GO,DFPROG
     39
     40  C
     41  C       SCALAR MONTE CARLO CODE TO TRANSPORT .001 TO 20.0 MEV
     42  C       GAMMA RAYS IN A CARBON CYLINDER OF LENGTH CL, RADIUS CRAD
     43  C
     44        REAL BANK(100,8),PBL(8)                                    BUGFIX
     45        INTEGER IBANK(100,2),IPBL(2)                               BUGFIX
     46        EQUIVALENCE (PBL(1),X),(PBL(2),Y),(PBL(3),Z),              BUGFIX
     47       +            (PBL(4),U),(PBL(5),V),(PBL(6),W),              BUGFIX
     48       +            (PBL(7),ERG),(PBL(8),WT),                      BUGFIX
     49       +            (IPBL(1),IA),(IPBL(2),NP)                      BUGFIX
     50        REAL BSCATI(35),TRANSI(35),ESCAPI(35)                      RENAME
     51        REAL BTOTI,TTOTI,ETOTI
     52        INTEGER INBNK,NBANK
     53        INTEGER KRN                                                ALG CH
     54        INTEGER CUTOFF
     55        IF(LPR(IPN).EQ.-1) RETURN
     56   7799 DFPROG=.FALSE.
```

```
      CALL AWRITE($DR(8),LAREAD($DR(8)))
      CALL AWRITE($DR(14),LAREAD($DR(14)))
      CALL AWRITE($DR(5),LAREAD($DR(5)))
      CALL AWRITE($DR(6),LAREAD($DR(6)))
      CALL AWRITE($DR(7),LAREAD($DR(7)))
      CALL AWRITE($DW(15),GO)
      GO=LAREAD($DW(15))
 7798 LPX(IPN)=LPX(IPN)+1
      LPR(IPN)=1
      LPR(IPCTXT)=LPR(IPCTXT)+1

C
            GO=LAREAD($DR(14))
C -- MAKE LOCAL COPY OF RANDOM SEED
      KRN=KERN
            GO=LAREAD($DW(14))
         DFPROG=.TRUE.
C -- MAKE LOCAL COPIES OF INITIAL SOURCE VALUES
      ERG=GIERG
      WT=GIWT
      U=GIU
      V=GIV
      W=GIW
      X=GIX
      Y=GIY
      Z=GIZ
      IA=IGIA
C
      NCOL=0
      NCO=0
      WCO=0
      WCP=0
      WRL=0.
      WRG=0.
      NR=0
      CUTOFF= 0
      ABSORB=0
      NS=0                                                    BUGFIX
      DO 5 I = 1,35
         BSCATI(I)=0.                                         BUGFIX
         TRANSI(I)=0.                                         BUGFIX
         ESCAPI(I)=0.                                         BUGFIX
    5 CONTINUE                                                STYLE
      BTOTI=0.                                                BUGFIX
      TTOTI=0.                                                BUGFIX
      ETOTI=0.                                                BUGFIX
C     INBNK=0                                                 EXTRA
C -- INITIALIZE LOCAL PARTICLE BANK INDEXES
      NBANK=0
      INBNK=0
C
C
C        CALCULATE DISTANCE DLS TO NEXT SURFACE INTERSECTION
C        FOR ALL THREE SURFACES AND ALSO THE NUMBER JA OF THE
C        NEXT SURFACE INTERSECTED
C
```

```
113      20 JA=0
114         CALL TRACK(IA,JA,X,Y,Z,U,V,W,CL,CL2,CRAD2,DLS)
115   C
116   C        FIND ENERGY POINTER FOR CROSS SECTIONS AND TALLYS
117   C
118         DO 30 IE = 1,35
119            IF(ERG.GT.E(IE)) GO TO 30
120            I = IE
121            GO TO 31
122      30 CONTINUE
123   C
124   C        INTERPOLATION TO GET CROSS SECTIONS AS F(ERG)
125   C
126      31 F=(ALOG(ERG)-EL(I-1))/(EL(I)-EL(I-1))
127         XSC=EXP( XC(I-1)+F*(XC(I)-XC(I-1)) )
128         XSPP=EXP( XPP(I-1)+F*(XPP(I)-XPP(I-1)) )
129         XSPE=EXP( XPE(I-1)+F*(XPE(I)-XPE(I-1)) )
130         XST  = XSC + XSPP + XSPE
131   C
132   C        CALCULATE DISTANCE TO NEXT COLLISION
133         S = -ALOG(RANF(KRN))/XST
134   C
135   C        SEE IF COLLISION IS STILL INSIDE CYLINDER
136   C        IF NOT, DO TALLYS; IF SO, DO COLLISION PHYSICS
137         IF(S.LT.DLS) GO TO 60
138         X=X+U*DLS
139         Y=Y+V*DLS
140         Z=Z+W*DLS
141         GO TO(42,50,53,52) JA
142      42 BSCATI(I) = BSCATI(I) + WT
143         BTOTI = BTOTI + WT
144         GO TO 11
145      52 TRANSI(I) = TRANSI(I) + WT
146         TTOTI = TTOTI + WT
147         GO TO 11
148      50 ESCAPI(I) = ESCAPI(I) + WT                                    RENAME
149         ETOTI = ETOTI + WT
150         GO TO 11
151   C  CROSS INTERNAL SURFACE  SPLIT OR ROULETTE
152      53 IAP=IA
153         IA=2-IA/2
154         T1=FIM(IA)/FIM(IAP)
155         IF(T1.GT.1.0) GO TO 57
156   C  RUSSIAN ROULETTE
157         IF(T1.LT.RANF(KRN)) GO TO 58
158         WTSAV=WT
159         WT=WT/T1
160         WRG=WRG+(WT-WTSAV)
161         GO TO 20
162   C  KILLED IN RUSSIAN ROULETTE
163      58 WRL=WRL+WT
164         NR=NR+1
165         GO TO 11
166   C  SPLITTING
167      57 NP=T1-1
168         WT=WT/T1
```

```
      NS=NS+NP
      NBANK=NBANK+NP
      INBNK=INBNK+1
      DO 59 IX=1,8                                                    BUGFIX
   59 BANK(INBNK,IX)=PBL(IX)
      DO 61 IX=1,2                                                    BUGFIX
   61 IBANK(INBNK,IX)=IPBL(IX)                                        BUGFIX
      GO TO 20
C  CHECK BANK BEFORE STARTING NEW PARTICLE
   11 IF(NBANK.EQ.0) GO TO 234
      DO 521 IX=1,8
  521 PBL(IX)=BANK(INBNK,IX)
      DO 522 IX=1,1
  522 IPBL(IX)=IBANK(INBNK,IX)
      NBANK=NBANK-1
      IBANK(INBNK,2)=IBANK(INBNK,2)-1                                 BUGFIX
      IF(IBANK(INBNK,2).EQ.0) INBNK=INBNK-1
      GO TO 20
C
C        COLLISIONS
   60 JA = 0
      X=X+U*S
      Y=Y+V*S
      Z=Z+W*S
      NCOL=NCOL+1
C  SURVIVAL BIAS
      WTSAV=WT
      WT=WT*(1.-XSPE/XST)
      ABSORB=ABSORB+(WTSAV-WT)
      XSTSB=XST-XSPE
C WEIGHT CUTOFF
      IF(WT.GT.WCP2) GO TO 832
      IF(WT*FIM(IA).LT.RANF(KRN)*WCP1*FIM(1)) GO TO 642
      WTSAV=WT
      WT=WCP1*FIM(1)/FIM(IA)
      WCP=WCP+(WT-WTSAV)
  832 CONTINUE
      IF(RANF(KRN).GE.XSC/XSTSB) GO TO 100
      T1 = 1.956917*ERG
C        GET NEW ENERGY T4 AND COMPTON SCATTERING ANGLE
      CALL KLEIN(T1,T4,KRN)                                           ARGS
      CSA = 1.+1./T1-1./T4
      T5 = .511008*T4
      IF(ABS(CSA).GT.1.) CSA=SIGN(1.,CSA)
      ERG = T5
C
C        SEE IF NEW ENERGY IS LESS THAN CUTOFF
      IF(ERG.GT.EC) GO TO 70
      CUTOFF = CUTOFF + 1
      GO TO 11
C        MAKE COMPTON ANGLE RELATIVE TO PROBLEM COORDINATE SYSTEM
   70 UOLD = U
      VOLD = V
      WOLD = W
      CALL ROTAS(CSA,KRN,U,V,W,UOLD,VOLD,WOLD)                        ARGS
      GO TO 20
```

```
      C
      C        PAIR PRODUCTION
  100 ERG = 0.511008
      WT = 2.*WT
      C
      C        CHECK ENERGY CUTOFF
      IF(ERG.GT.EC) GO TO 110
      CUTOFF = CUTOFF + 1
      GO TO 11
      C
      C        ISOTROPIC EMISSION IN LAB SYSTEM
  110 CALL ISOS(U,V,W,KRN)                                          ARGS
      GO TO 20
      C
      C        PHOTOELECTRIC ABSORPTION
      C NOW HANDLED BY SURVIVAL BIASING
      C 130 ABSORB = ABSORB + WT
      C     GO TO 11
      C  TERMINATE PARTICLE TO WEIGHT CUTOFF
  642 WCO=WCO+WT
      NCO=NCO+1
      GO TO 11
      C -- GET EXCLUSIVE UPDATE ACCESS TO RUN STATISTICS
  234       GO=LAREAD($DR(8))
      IGNCOL=IGNCOL+NCOL
      IGNCO=IGNCO+NCO
      GWCO=GWCO+WCO
      GWCP=GWCP+WCP
      GWRL=GWRL+WRL
      GWRG=GWRG+WRG
      IGNR=IGNR+NR
      IGCUTF=IGCUTF+CUTOFF
      GABSOR=GABSOR+ABSORB
      IGNS=IGNS+NS
      DO 829 I=1,35
      BSCAT(I)=BSCAT(I)+BSCATI(I)
      BSCAT2(I)=BSCAT2(I)+BSCATI(I)**2
      TRANS(I)=TRANS(I)+TRANSI(I)
      TRANS2(I)=TRANS2(I)+TRANSI(I)**2
      ESCAPE(I)=ESCAPE(I)+ESCAPI(I)                                 RENAME
      ESCAP2(I)=ESCAP2(I)+ESCAPI(I)**2                              RENAME
  829 CONTINUE
      BTOT=BTOT+BTOTI
      TTOT=TTOT+TTOTI
      ETOT=ETOT+ETOTI
      BTOT2=BTOT2+BTOTI**2
      TTOT2=TTOT2+TTOTI**2
      ETOT2=ETOT2+ETOTI**2
      C -- GIVE BACK UPDATE ACCESS TO RUN STATISTICS
            CALL AWRITE($DR(8),.FALSE.)
            DFPROG=.TRUE.
      C -- SIGNAL COMPLETION OF THIS PARTICLE HISTORY
            CALL AWRITE($DW(15),.FALSE.)
            DFPROG=.TRUE.
            CALL AWRITE($DR(15),.FALSE.)
            DFPROG=.TRUE.
```

```
C       GO TO 10                                                        OLD
              GO TO  7797
 7797 IF(.NOT.DFPROG) THEN
      GO=LAREAD($LTR)
      CALL PTRACE(IPN,48,0,LPX(IPN),0.0)
      CALL AWRITE($LTR,VALUE($LTR))
         LPR(IPN)= -1
         RETURN
      ENDIF
      GO TO  7799
      END
C==>END:p17.f
      SUBROUTINE TRACK(IA,JA,X,Y,Z,U,V,W,CL,CL2,CRAD2,DLS)              ARGS
C        CALCULATE ALL INTERSECTIONS WITH ALL THREE SURFACES
      DLSS = 1.0E10
      IF(IA.EQ.2) GO TO 19
      DO 300 J=1,3
      D1 = -1.0
      GO TO (55,160,50),J
   50 IF(V.EQ.0.) GO TO 300
      D1 = (CL-Y)/V
      GO TO 280
   55 IF(V.EQ.0.) GO TO 300
      D1 = -Y/V
      GO TO 280
  160 T1 = U**2 + W**2
      IF(T1.EQ.0.) GO TO 300
      A1 = (X*U + Z*W)/T1
      B1 = (X**2 + Z**2 - CRAD2)/T1
      T1 = A1**2 - B1
      IF(T1.LT.0.) GO TO 300
      T2 = SQRT(T1)
      D1 = -A1 + T2
      D2 = -A1 - T2
C     IF(J.EQ.JA) D2=D1=-2.*A1                                          OLD
      IF(J.EQ.JA) THEN                                                  NEW
         D1=-2.*A1                                                      NEW
         D2=-2.*A1                                                      NEW
      ENDIF                                                             NEW
      GO TO 290
  280 D2 = -D1
  290 IF(D1.LE.0.) GO TO 300
      IF(D2.GT.0.) D1=D2
      IF(D1.GE.DLSS) GO TO 300
      JAS = J
      DLSS = D1
  300 CONTINUE
      DLS = DLSS+1.0E-10
      JA = JAS
      RETURN
   19 DO 301 J=2,4
      D1 = -1.0
      GO TO (56,161,51,56),J
   51 IF(V.EQ.0.) GO TO 301
      D1 = (CL-Y)/V
      GO TO 281
```

```
      56 IF(V.EQ.0.) GO TO 301
         D1 = (CL2-Y)/V
         GO TO 281
     161 T1 = U**2 + W**2
         IF(T1.EQ.0.) GO TO 301
         A1 = (X*U + Z*W)/T1
         B1 = (X**2 + Z**2 - CRAD2)/T1
         T1 = A1**2 - B1
         IF(T1.LT.0.) GO TO 301
         T2 = SQRT(T1)
         D1 = -A1 + T2
         D2 = -A1 - T2
C        IF (J.EQ.JA) D2=D1=-2.*A1                                    OLD
         IF (J.EQ.JA) THEN                                            NEW
             D1=-2.*A1                                                NEW
             D2=-2.*A1                                                NEW
         END IF                                                       NEW
         GO TO 291
     281 D2 = -D1
     291 IF(D1.LE.0.) GO TO 301
         IF(D2.GT.0.) D1=D2
         IF(D1.GE.DLSS) GO TO 301
         JAS = J
         DLSS = D1
     301 CONTINUE
         DLS = DLSS+1.0E-10
         JA = JAS
         RETURN
         END

         SUBROUTINE KLEIN(T1,T4,KRN)                                  ARG
C           SAMPLE FROM KLEIN-NISHINA USING INVERSE FIT.
C           T1=ENERGY IN, T4=ENERGY OUT, IN UNITS OF THE REST MASS
C           OF AN ELECTRON.
C
         RN=RANF(KRN)
         T2=1./T1
         T4=2.*T1+1.
         T5=1./T4
         T6=ALOG(T4)
         T3=2.*T1*(1.+T1)*T5**2+4.*T2+(1.-2.*T2*(1.+T2))*T6
         IF(T1.LE.1.16666667)GO TO 20
         T7=1.65898+T2*(.62537*T2-1.00796)
         T3=T7/T3
         IF(RN.LE.T3)GO TO 10
         T4=(T6-1.20397)/(1.-T3)
         T7=.3*EXP(T4*(T3-RN))
         GO TO 40
      10 T4=T7/(3.63333+T2*(5.44444*T2-4.66667))
         T7=.5*T7
         T2=RN/T3
         T3=2.1
         T5=1.4
         GO TO 30
      20 T4=T3/(T4+T5)
         T7=.5*T3
```

```
      T2=RN
      T5=1.-T5
      T3=3.*T5
      T5=2.*T5
   30 T7=1.+T2*(T2*(2.*T7+T4-T3+T2*(T5-T7-T4))-T7)
   40 T4=T7*T1
      RETURN
      END

      SUBROUTINE ISOS(U,V,W,KRN)                                        ARGS
C        SAMPLE A DIRECTION U,V,W ISOTROPICALLY.
C
   10 T1=2.*RANF(KRN)-1.
      T2=2.*RANF(KRN)-1.
      RSQ=T1**2+T2**2
      IF(RSQ.GT.1.0)GO TO 10
      U=2.*RSQ-1.
      T3=SQRT((1.-U**2)/RSQ)
      V=T1*T3
      W=T2*T3
      RETURN
      END

      SUBROUTINE ROTAS(C,KRN,U,V,W,UOLD,VOLD,WOLD)                      ARGS
C        ROTATE UOLD,VOLD,WOLD TO U,V,W THROUGH A POLAR
C        ANGLE WHOSE COSINE IS C, AND THROUGH AN AZIMUTHAL
C        ANGLE SAMPLED UNIFORMLY.
C
   10 T1=2.*RANF(KRN)-1.
      T2=2.*RANF(KRN)-1.
      R=T1**2+T2**2
      IF(R.GT.1.0)GO TO 10
      R=SQRT((1.-C**2)/R)
      T1=T1*R
      T2=T2*R
      IF(ABS(WOLD).GT..999999)GO TO 30
      S=SQRT(1.-WOLD**2)
      U=UOLD*C+(T1*UOLD*WOLD-T2*VOLD)/S
      V=VOLD*C+(T1*VOLD*WOLD+T2*UOLD)/S
      W=WOLD*C-T1*S
      RETURN
   30 U=T1
      V=T2
      W=WOLD*C
      RETURN
      END

      REAL FUNCTION RANF(KERN)                                          ALG CH
      KERN = MOD(1+9621*KERN,131072)                                    ALG CH
      RANF = FLOAT(KERN)/131072.                                        ALG CH
      RETURN                                                            ALG CH
      END                                                               ALG CH
```

```
      C--<<<p18.f>>>
      C
      C----- CKCOMP -- check for run completion
      C
            SUBROUTINE P18(IPN,IPCTXT)
      C
      C
      C---------------- (HEPUNIX FORTRAN SYSTEM TABLES) ----------------
            COMMON /SYSTAB/ $LTR,$DW,$DR,LPR(40),LPS(40),LPX(40),LNL(40)
            LOGICAL $LTR,$DW(15),$DR(15)
      C
      C-> d06: run stats  - run statistics
            COMMON /D06/ IGNCOL,IGNCO,GWCO,GWCP,GWRL,IGNR,GWRG,IGCUTF,
           +   GABSOR,IGNS,
           +   TRANS,ESCAPE,BSCAT,TRANS2,ESCAP2,BSCAT2,
           +   BTOT2,TTOT2,ETOT2,ETOT,BTOT,TTOT
            INTEGER IGNCOL,IGNCO,IGNR,IGCUTF,IGNS
            REAL GWCO,GWCP,GWRL,GWRG,GABSOR,
           +   TRANS(35),ESCAPE(35),BSCAT(35),
           +   TRANS2(35),ESCAP2(35),BSCAT2(35),
           +   BTOT2,TTOT2,ETOT2,ETOT,BTOT,TTOT
      C-> d11: NPPC;NPC   - particle completion control
            COMMON /D11/ NPC,NPPC
            INTEGER NPC,NPPC
      C-> d14: signal     - particle history completion signal
            COMMON /D14/ ID14
            INTEGER ID14
            LOGICAL GO,DFPROG
      
            IF(LPR(IPN).EQ. -1) RETURN
      7799  DFPROG=.FALSE.
            IF(LNL(IPN).GT.0) THEN
               IF(FULL($DR(8))) THEN
                  CALL AWRITE($DW(13),GO)
                  GO=LAREAD($DW(13))
                  GO=LAREAD($DR(8))
                  GO=LAREAD($DW(8))
                  DFPROG=.TRUE.
                        LNL(IPN)=LNL(IPN)-1
               ENDIF
               IF(LNL(IPN).NE.0) THEN
            GO=LAREAD($LTR)
            CALL PTRACE(IPN,71,LNL(IPN),LPX(IPN),0.0)
            CALL AWRITE($LTR,VALUE($LTR))
                  DFPROG=.FALSE.
               ENDIF
               GO TO  7797
            ENDIF
            CALL AWRITE($DR(8),LAREAD($DR(8)))
            CALL AWRITE($DR(11),LAREAD($DR(11)))
            CALL AWRITE($DR(15),LAREAD($DR(15)))
            CALL AWRITE($DR(8),LAREAD($DR(8)))
            CALL AWRITE($DW(13),GO)
            GO=LAREAD($DW(13))
      7798  LPX(IPN)=LPX(IPN)+1
            LPR(IPN)=1
```

```
      LPR(IPCTXT)=LPR(IPCTXT)+1

C
            GO=LAREAD($DR(15))
            GO=LAREAD($DW(15))
         DFPROG=.TRUE.
      NPC=NPC+1
      IF (NPC.GE.NPPC) THEN
            CALL AWRITE($DW(13),.FALSE.)
         DFPROG=.TRUE.
            CALL AWRITE($DR(13),.FALSE.)
         DFPROG=.TRUE.
            LNL(IPN)=LNL(IPN)+1
            GO=LAREAD($DR(11))
            GO=LAREAD($DW(11))
         DFPROG=.TRUE.
      ENDIF
            GO TO  7797
 7797 IF(.NOT.DFPROG) THEN
      GO=LAREAD($LTR)
      CALL PTRACE(IPN,48,0,LPX(IPN),0,0)
      CALL AWRITE($LTR,VALUE($LTR))
         LPR(IPN)= -1
         RETURN
      ENDIF
      GO TO  7799
      END
C==>END:p18.f
```

APPENDIX F

Reference Output for 100 Particles

Apr 30 14:15 1985 reference.100 Page 1

NCOL 117
SCALERT

NPS = 100

E	BSCAT	REL ERROR
1.000e-03	0. e+00	0.
1.500e-03	0. e+00	0.
2.000e-03	0. e+00	0.
3.000e-03	0. e+00	0.
4.000e-03	0. e+00	0.
5.000e-03	0. e+00	0.
6.000e-03	0. e+00	0.
8.000e-03	0. e+00	0.
1.000e-02	0. e+00	0.
1.500e-02	0. e+00	0.
2.000e-02	0. e+00	0.
3.000e-02	0. e+00	0.
4.000e-02	0. e+00	0.
5.000e-02	0. e+00	0.
6.000e-02	0. e+00	0.
8.000e-02	0. e+00	0.
1.000e-01	0. e+00	0.
1.500e-01	0. e+00	0.
2.000e-01	0. e+00	0.
3.000e-01	0. e+00	0.
4.000e-01	0. e+00	0.
5.000e-01	0. e+00	0.
6.000e-01	0. e+00	0.
8.000e-01	0. e+00	0.
1.000e+00	0. e+00	0.
1.500e+00	0. e+00	0.
2.000e+00	0. e+00	0.
3.000e+00	0. e+00	0.
4.000e+00	0. e+00	0.
5.000e+00	0. e+00	0.
6.000e+00	0. e+00	0.
8.000e+00	0. e+00	0.
1.000e+01	0. e+00	0.
1.500e+01	0. e+00	0.
2.000e+01	0. e+00	0.
TOTAL	0 e+00	0

E	ESCAPE	REL ERROR
1.000e-03	0. e+00	0.
1.500e-03	0. e+00	0.
2.000e-03	0. e+00	0.
3.000e-03	0. e+00	0.
4.000e-03	0. e+00	0.
5.000e-03	0. e+00	0.
6.000e-03	0. e+00	0.

8.000e-03	0. e+00	0.
1.000e-02	0. e+00	0.
1.500e-02	0. e+00	0.
2.000e-02	0. e+00	0.
3.000e-02	0. e+00	0.
4.000e-02	0. e+00	0.
5.000e-02	0. e+00	0.
6.000e-02	0. e+00	0.
8.000e-02	0. e+00	0.
1.000e-01	9.949e-03	0.995
1.500e-01	0. e+00	0.
2.000e-01	0. e+00	0.
3.000e-01	4.000e-02	0.490
4.000e-01	8.500e-02	0.344
5.000e-01	6.000e-02	0.339
6.000e-01	1.700e-01	0.273
8.000e-01	7.500e-02	0.332
1.000e+00	5.500e-02	0.404
1.500e+00	5.500e-02	0.404
2.000e+00	4.500e-02	0.447
3.000e+00	6.500e-02	0.355
4.000e+00	8.000e-02	0.327
5.000e+00	6.500e-02	0.355
6.000e+00	0. e+00	0.
8.000e+00	0. e+00	0.
1.000e+01	0. e+00	0.
1.500e+01	0. e+00	0.
2.000e+01	0. e+00	0.
TOTAL	8.049e-01	0.0589

E	TRANS	REL ERROR
1.000e-03	0. e+00	0.
1.500e-03	0. e+00	0.
2.000e-03	0. e+00	0.
3.000e-03	0. e+00	0.
4.000e-03	0. e+00	0.
5.000e-03	0. e+00	0.
6.000e-03	0. e+00	0.
8.000e-03	0. e+00	0.
1.000e-02	0. e+00	0.
1.500e-02	0. e+00	0.
2.000e-02	0. e+00	0.
3.000e-02	0. e+00	0.
4.000e-02	0. e+00	0.
5.000e-02	0. e+00	0.
6.000e-02	0. e+00	0.
8.000e-02	0. e+00	0.
1.000e-01	0. e+00	0.
1.500e-01	0. e+00	0.
2.000e-01	0. e+00	0.
3.000e-01	0. e+00	0.
4.000e-01	0. e+00	0.
5.000e-01	0. e+00	0.

6.000e-01	0.	e+00	0.
8.000e-01	0.	e+00	0.
1.000e+00	0.	e+00	0.
1.500e+00	0.	e+00	0.
2.000e+00	0.	e+00	0.
3.000e+00	5.000e-03		0.995
4.000e+00	0.	e+00	0.
5.000e+00	1.000e-02		0.700
6.000e+00	2.500e-01		0.143
8.000e+00	0.	e+00	0.
1.000e+01	0.	e+00	0.
1.500e+01	0.	e+00	0.
2.000e+01	0.	e+00	0.
TOTAL	2.650e-01		0.1398

ABSORB = 5.101e-05 CUTOFF = 0
TRACKS CREATED BY SPLITTING 45 TRACKS LOST TO ROULETTE 0
WEIGHT CREATED BY ROULETTE 5.0000e-03 WEIGHT LOST TO ROULETTE 0. e+00
TRACKS LOST TO WEIGHT CUTOFF 0
WEIGHT CREATED BY WEIGHT CUTOFF 0. e+00 WEIGHT LOST TO WEIGHT CUTOFF 0. e+00

TOTAL TIME = 0. e+00 SECONDS