Parallel processing on the DENECLOR HEP with large grain data flow techniques

Robert G. II Babb
PARALLEL PROCESSING ON THE DENELCOR HEP WITH LARGE GRAIN DATA FLOW TECHNIQUES

Robert G. Babb II
Lise Store
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.
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
  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)
  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
```

was changed to:

```
NPP=100
```

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.O.) XPP(I)=1.OE-123
```

was changed to:

```
IF (XPP(I).EQ.O.) XPP(I)=1.OE-37
```

(see also [original-39] and [f77-59]).
7) A real variable was initialized with an integer constant:

\[ \text{FIM}(1) = 1 \]

[original-72]

and for stylistic reasons was changed:

\[ \text{FIM}(1) = 1.0 \]

[f77-103]

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

\[ \text{INBNK} = 0 \]

[original-74]

changed to:

\[ \text{INBNK} = 0 \]

[f77-106]

9) The real-integer equivalencing problem related to banking particle values was fixed. In the original version of GAME-TEB, 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):

\[ \text{COMMON X,Y,Z,U,V,W,ERG,IA,WT,NP} \]

\[ \text{DIMENSION BANK(100,10),PBL(10),IBANK(100,10)} \]

\[ \text{EQUIVALENCE (PBL,X),(BANK,IBANK)} \]

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:

```fortran
COMMON X,Y,Z,U,V,W,ERG,WT,IA,NP
DIMENSION BANK(100,8),PBL(8),IBANK(100,2),IPBL(2)
EQUIVALENCE (PBL,X), (IPBL,IA)
```

(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 (!):

```fortran
T1=2.*RANF(KRN)-1.
T2=2.*RANF(KRN)-1.
RSQ=T1**2+T2**2
U=2.*RSQ-1.
T3=SQR((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)
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:

\[ S = -\text{ALOG}(\text{RANF}(\text{KRN}))/\text{XST} \]  

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
```

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:

```plaintext
REAL FUNCTION RANDO(KERN)  
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 E).

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:

\[ \text{p10-(set up problem constants)} \]

[77-53 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
C INITIALIZE PROBLEM INPUT
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 pl2- (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 pl2.

The process pl3- (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)
- d10- (particle count for report)

The LGDF process p13 also initializes the value (KRN2 on d12) used to seed the first random number generator (RAND0 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 [77-294 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:
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 pO2-(run particle histories).
The bulk of the original GAMEB 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 GAMEB 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 \[lgdf\text{-}pl7.m \text{ 228}2\]:

```
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:

\[
\text{IF} \ (\text{NPC} < \text{NPPC}) \ \text{THEN} \quad [\lgdf\text{-}p18.m \text{ 5}]
\]

was changed to:

\[
\text{IF} \ (\text{NPC} \geq \text{NPPC}) \ \text{THEN}
\]

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 dl3 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 dl3 did not agree with the name used in the actual code.

Several minor corrections were necessary:
   a) In [lgdf-dl3] 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 do6 [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 des-
cendants) 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($DW(5),GO)
CALL AWRITE($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 pl4 at [lgdf-pl4.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


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)
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 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.
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

![Diagram](a) update

![Diagram](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.
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, consistent² 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 UNIX³ 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

²A 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.

³UNIX is a trademark of Bell Laboratories.
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. 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.


---

*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

![Diagram of parallel triangular solver]

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 immediately 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.
MMwork p02
[p02]

Fig. 10. LGDF process network for MMWORK (p02).
Fig. 11. Data path and program definitions for the parallel triangular solver example.
Fig. 12. Encoded data path/program connectivity in the Wirelist File.
Fig. 13. Data declaration files for the parallel triangular solver example.

```fortran
DO1:
T, N, K, M
   REAL T(15, 15)
   INTEGER N, K, M

DO2:
Y
   REAL Y(15)

DO3:
NMNSUM
   INTEGER NMNSUM(15)

DO4:
ITS, KTS, NTS
   INTEGER ITS, KTS, NTS

DO5:
NRBI
   INTEGER NRBI

DO6:
IMM, JMM, KMM, NMM
   INTEGER IMM, JMM, KMM, NMM

DO7:
NRBC
   INTEGER NRBC

DO8:
MMCI
   INTEGER MMCI
```

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

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_(do2)
   set_(do1)
C--INITIALIZE THE ROW BLOCK COMPLETION COUNTS
   DO 50 I=1, K
       NMNSUM(I)=0
   50 CONTINUE
   set_(do3)
C--INITIALIZE TS WORK VARIABLES
   ITS=1
   KTS=N- (M-1)*K
   NTS=1
   set_(do4)
end_(p10)
program_(p11,[p01])
begin_
states_(s00,s01,s02)
state_(s00,issue first TS work request)
set_(d04)
C--(NOTE: NO SET OF d05 ROW BLOCK INTERLOCK)
C--CHECK FOR LAST TRIANGLE
IF (NTS.EQ.M) THEN
  next_(s02,set result vector)
ELSE
  next_(s01,issue next TS work)
ENDIF
suspend_
state_(s01,issue next TS work)
ITS=ITS+KTS
KTS=K
set_(d04)
C--SET ROW BLOCK INTERLOCK
NRBI=NTS
set_(d05)
C--CHECK FOR LAST TRIANGLE
NTS=NTS+1
IF (NTS.EQ.M) THEN
  next_(s02,set result vector)
ENDIF
suspend_
state_(s02,set result vector)
set_(d02)
WRITE(6,40) (Y(I),I=1,N)
40 FORMAT('Y=',15F5.2)
clear_(d01)
clear_(d03)
clear_(d04)
suspend_
end_(p11)

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

program_(p12,[p01])
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_(d04)
C-- ENABLE MMWORK
    IMM=ITS+KTS
    JMM=ITS
    KMM=KTS
    NMM=NTS+1
    set_(d06)
ENDIF
suspend_
end_(p12)

Fig. 14(c). An LGDF program for TS (p12).
Fig. 14(a). An LGDF program for WAITRB (p13).

```
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_([00, [01]
  state_([00, issue first MM work request)
    set_(d06)
    next_([01, issue next work request)
    suspend_
  state_([01, issue next work request)
    IMM=IMM+X
    IF (IMM.GT.N) THEN
      next_([00, issue first MM work request)
      clear_(d06)
    ELSE
      IMM=IMM+1
      set_(d06)
    ENDIF
    suspend_
end_(p14)
```

Fig. 14(e). An LGDF program for MMWORK (p14).
program (p15, [p02], *)
C--LOCAL VARIABLES
   REAL Y(HAT(15), TEMP
   INTEGER LI, LJ, LX, LN, I, J, II
begin
C--WAIT FOR EXCLUSIVE READ ACCESS TO INPUT ARGS
read_ (do6)
C--MAKE LOCAL COPIES OF INPUT ARGS
   LI=IM
   LJ=JM
   LX=KM
   LN=NM
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
   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)
   MMCM=LN
   aset_ (do8)
suspend_
end_ (p15)

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

program (p16, [p02])
C--LOCAL VARIABLE
   INTEGER LNRB
begin
C--MAKE LOCAL COPY OF INPUT ARG
   LNRB=MMCM
   aclear_ (do8)
   NMMSUM(LNRB)=NMMSUM(LNRB)+1
C-- CHECK FOR ALL MM's CHECKED IN THIS ROW BLOCK
   IF (NMMSUM(LNRB)+1) .EQ. LNRB) THEN
      NRB=NRB
      aset_ (do7)
   ENDIF
   suspend_
end_ (p16)

Fig. 14(g). An LGDF program for MMCKIN (p16).
C---- TS -- solve triangular matrix
SUBROUTINE P12(IPN,IPCTX)

C---- (HEP FORTRAN SYSTEM TABLES) ---------------
COMMON /SYSTAB/ $LTR, $DR, LPR(16), LPX(16), LNL(16)
LOGICAL $LTR, $DR(11), LPR(11)
LOGICAL GO, DFPROG
C-> d01: TS problem - (T(15,15),N,K,M)
COMMON /DO1/ T,N,K,M
REAL T(15,15)
INTEGER N,K,M
C-> d02: Y - result vector (space)
COMMON /DO2/ Y
REAL Y(15)
C-> d04: TS control - (ITS,KTS,NTS)
COMMON /DO4/ ITS,KTS,NTS
INTEGER ITS,KTS,NTS
C-> d06: MM control - (IBM,JMM,MMM,MMM)
COMMON /DO6/ IBM,JMM,MMM,MMM
INTEGER IBM,JMM,MMM,MMM

IF (LPR(IPN).EQ.-1) RETURN
DEFPROG=.FALSE.
$DR(3)=$DR(3)
$DR(4)=$DR(4)
$DR(6)=$DR(6)
$DR(8)=.FALSE.
GO=$DR(8)

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
20 CONTINUE

IF (NTS.LT.M) THEN
   C -- ENABLE WORK
   IBM=ITS+KTS
   JMM=ITS
   KMM=KTS
   MMM=NTS-1
   $DR(8)=.FALSE.
   DFPROG=.TRUE.
   $DR(8)=.FALSE.
   DFPROG=.TRUE.
ENDIF

IF (.NOT.DEFPROG) THEN
   GO=$LTR
   CALL PTRACE(IPN,48.0,LPX(IPN),.O.O)
   $LTR=VALUE($LTR)
   LPR(IPN)=-1
   RETURN
ENDIF

GO TO 7799
END

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


APPENDIX B

[original]

Original (CDC?) Sequential GAMTEB

B-1
C HERE COMES Sgamte (SEQUENTIAL VERSION --- THE ORIGINAL)

PROGRAM GAMTEB (OUTPUT, TAPE4=OUTPUT)

SCALAR MONTE CARLO CODE TO TRANSPORT .001 TO 20.0 MEV

GAMMA RAYS IN A CARBON CYLINDER OF LENGTH CL, RADIUS CRAD

COMMON X, Y, Z, U, V, W, ERG, IA, WT, NP, VOLD, WOLD, CL, 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), RESCAP(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, .006, .01, .015, .02, .03, .04, .05, .06, .08,

2 .09, .112, .3, .4, .5, 6, 8, 1, 1.5, 2, 3, 4, 5,

3 6, 8, 10, 15, 20.

DATA (XC(I),I=1,35)/.015, .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, .014

DATA (XPP(I),I=1,35)/0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

1 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.000792,

2 0.000316, .000923, .0013, .00208, .00256, .00343,

3 .00414, .00547, .00652/

DATA (XPE(I),I=1,35)/.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, .014

DATA (XC(35),XPP(35))/0.80, 1.877, 3.7, 3.87, 3.97,

1 4.01, 4.01, 4.01, 4.01, 4.01, 4.01, 4.01, 4.01, 4.01,

2 0.0181, .0181, .0181, .0181, .0181, .0181, .0181, .0181

3 0.00481, .00179, .000862, .000234, .0000918

4 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

CONVERT CROSS-SECTION UNITS TO BE CM.

DO 1 I=1,35

XC(I)=ALOG(XC(I)*RHO)

1 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 INITIALIZE PROBLEM INPUT

NPP=500000

WCP=25

EC = .001

NPS = 0

KRN = 1234542321

CL=20.0
CL2=CL+10
CRAD=1.0
CRAD2=CRAD*2
CUTOFF = 0
BTOT=0.0
BTOT2=0.0
TQT=0.0
ETOT=0.0
ETOT2=0.0
WRL=0.0
WRG=0.0
NR=0
INBNK=0
NBANK=0
FIM(1)=1
FIM(2)=2.0
INBNK=0
DO 3 I = 1, 35
TRANS2(I)= 0.0
BSCAT2(I)= 0.0
ESCAPE2(I)= 0.0
TRANS(I)= 0.0
BSCAT(I)= 0.0
ESCAPE(I)= 0.0
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 CALCULATE DISTANCE DLS TO NEXT SURFACE INTERSECTION FOR ALL THREE SURFACES AND ALSO THE NUMBER JA OF THE NEXT SURFACE INTERSECTED
JA=0
CALL TRACK
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
INTERPOLATION TO GET CROSS SECTIONS AS F(ERG)

\[ F = \frac{\text{ALOG}(\text{ERG}) - \text{EL}(I-1))}{\text{EL}(I) - \text{EL}(I-1))} \]

\[ \text{XSC} = \exp(\text{XC}(I-1) + F \cdot (\text{XC}(I) - \text{XC}(I-1))) \]
\[ \text{XPP} = \exp(\text{XP}(I-1) + F \cdot (\text{XP}(I) - \text{XP}(I-1))) \]
\[ \text{XPE} = \exp(\text{XE}(I-1) + F \cdot (\text{XE}(I) - \text{XE}(I-1))) \]
\[ \text{XST} = \text{XSC} + \text{XPP} + \text{XPE} \]

CALCULATE DISTANCE TO NEXT COLLISION

\[ S = -\text{ALOG}(\text{RANF(KRN)})/\text{XST} \]

SEE IF COLLISION IS STILL INSIDE CYLINDER

IF NOT, DO TALLYS; IF SO, DO COLLISION PHYSICS

\[ F = \frac{(\text{ALOG}(\text{ERG}) - \text{EL}(I-1))}{\text{EL}(I) - \text{EL}(I-1))} \]

\[ \text{XSC} = \exp(\text{XC}(I-1) + F \cdot (\text{XC}(I) - \text{XC}(I-1))) \]
\[ \text{XPP} = \exp(\text{XP}(I-1) + F \cdot (\text{XP}(I) - \text{XP}(I-1))) \]
\[ \text{XPE} = \exp(\text{XE}(I-1) + F \cdot (\text{XE}(I) - \text{XE}(I-1))) \]
\[ \text{XST} = \text{XSC} + \text{XPP} + \text{XPE} \]

53 IAP = IA

IF (T1, GT, 1.0) GO TO 57

RUSSIAN ROULETTE

IF (T1, LT, RANF(KRN)) GO TO 50

KILLED IN RUSSIAN ROULETTE

SPLITTING

IF (NBANK, EQ, 0) GO TO 234

CHECK BANK BEFORE STARTING NEW PARTICLE

GO TO 20
169 IF(IBANK(INBNK,10).EQ.0) INBNK=INBNK-1
170 GO TO 20
171 C COLLISIONS
172 C 60 JA = 0
173 X=X+U*W
174 Y=Y+V*W
175 Z=Z+W*W
176 C NCOL=NCOL+1
177 C SURVIVAL BIAS
178 WTSAV=WT
179 WT=WT*(1.-XSPE/XST)
180 XSTB=XST-XSPE
181 ABSORB=ABSORB+(WTSAV-WT)
182 C WEIGHT CUTOFF
183 IF(WT.GT.WCP2) GO TO 832
184 IF(WT*FIM(IA).LT.RANF(KRN)+WCP1*FIM(1)) GO TO 642
185 WTSAV=WT
186 WT=WCP1*FIM(1)/FIM(IA)
187 WCP=WCP-(WT-WTSAV)
188 CONTINUE
189 B32 GO TO 100
190 IF(RANF(KRN).GE.XSC/XSTSB) GO TO 100
191 T1 = 1.956917*ERG
192 C GET NEW ENERGY T4 AND COMPTON SCATTERING ANGLE
193 CALL KLEIN(T1,T4)
194 CSA = 1.+1./T1-1.1/T4
195 T5 = .511008*T4
196 IF(ABS(CSA).GT.1.) CSA=SIGN(1.,CSA)
197 ERG = T5
198 C SEE IF NEW ENERGY IS LESS THAN CUTOFF
199 IF(ERG.GT.EC) GO TO 110
200 CUTOFF = CUTOFF + 1
201 GO TO 11
202 C MAKE COMPTON ANGLE RELATIVE TO PROBLEM COORDINATE SYSTEM
203 CALL ROTAS(CSA)
204 VOLD = U
205 WOLD = V
206 CALL ROTAS(CSA)
207 GO TO 20
208 C PAIR PRODUCTION
209 C CHECK ENERGY CUTOFF
210 IF(ERG.GT.EC) GO TO 110
211 CUTOFF = CUTOFF + 1
212 GO TO 11
213 C ISOTROPIC EMISSION IN LAB SYSTEM
214 C PHOTONELECTRIC ABSORPTION
215 C NOW HANDLED BY SURVIVAL BIASING
216 CALL ISOS
217 GO TO 20
218 C PHOTON ELECTRIC ABSORPTION
219 C NOW HANDLED BY SURVIVAL BIASING
220 GO TO 20
C 130 ABSORB = ABSORB + WT
C GO TO 11
C TERMINATE PARTICLE TO WEIGHT CUTOFF
642 NCD = NCD + WT
NCD = NCD + 1
GO TO 11
234 DO 829 I = 1, 35
BSCAT(I) = BSCAT(I) + BSCAT(ID)
BSCAT2(I) = BSCAT2(I) + BSCAT2(ID)**2
TRANS(I) = TRANS(I) + TRANSID
TRANS2(I) = TRANS2(I) + TRANSID**2
ESCAPE(I) = ESCAPE(I) + ESCAPEID
ESCAPE2(I) = ESCAPE2(I) + ESCAPEID**2
BSCATI(I) = 0.
TRANSI(I) = 0.
ESCAPEI(I) = 0.
829 CONTINUE
BTOT = BTOT + BTOTI
TDTI = TDTI + TDTID
BTOT2 = BTOT2 + BTOTI**2
TDTI2 = TDTI2 + TDTID**2
ETOTI = ETOTI + ETOTID
ETOT2 = ETOT2 + ETOTI**2
BTOTI = 0.
ETOTI = 0.
GO TO 10
C PRINT OUTPUT
140 NPS = NPS - 1
CALL SECOND(I)
TEND = T1 - TO
WRITE(4, 7634) NCDL
7634 FORMAT(9H NCDL, 110)
WRITE(4, 1401) NPS
1401 FORMAT(7HBSCAT, /)
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
DO 203 RTRANS(I) = SQRT((TRANS2(I) - TRANS(I)**2)/ RNPS)
RTRANS(I) = RTRANS(I)/TRANS(I)
IF (BSCAT(I).NE.0.0) GO TO 205
RBSCAT(I) = 0.0
GO TO 206
205  RBCAT(I) = SQRT((BSCAT2(I) - BSCAT(I)**2) / RNSP)
206  RBSCAT(I) = BSCAT(I) / BSCAT(I)
207  IF(ECAPE(I).NE.0.0) GO TO 207
208  ESCAPE(I) = 0.0
209  GO TO 209
210  RESCAPE(I) = SQRT((ESCAPE2(I) - ESCAPE(I)**2) / RNSP)
211  RESCAPE(I) = RESCAPE(I) / ESCAPE(I)
212
213  WRITE(4,210) E(I), BSCAT(I), RBCAT(I)
214  CONTINUE
215  TTOT = TTOT/RNSP
216  BTOT2 = BTOT2/RNSP
217  BTOT2 = BTOT2/RNSP
218  ETOT = ETOT/RNSP
219  IF(TTOT.NE.0.0) GO TO 2000
220  RTTOT = 0.0
221  GO TO 2001
222  RTTOT = SQRT((TTOT - TTOT**)2/RNSP)
223  RRTOT = RRTOT/TTOT
224  IF(BTOT.NE.0.0) GO TO 2002
225  RBOTOT = 0.0
226  GO TO 2003
227  RBOTOT = SQRT((BTOT2 - BTOT**)2/RNSP)
228  RBTOT = RBTOT/BTOT
229  IF(EOT.UE.0.0) GO TO 2004
230  RETOT = 0.0
231  GO TO 2005
232  RETOT = SQRT((ETOT2 - ETOT**)2/RNSP)
233  RETOT = RETOT/ETOT
234  CONTINUE
235  WRITE(4,2020) BTOT, RBOT
236  WRITE(4,201) /
237  WRITE(4,201) /
238  201 FORMAT(///, 8X, 1HE.13X, 6HESCAPE, 8X, 9HREL ERROR)
239  DD 223 I=1.35
240  WRITE(4,210) E(I), ESCAPE(I), RESCAPE(I)
241  CONTINUE
242  WRITE(4,202) ETOT, RETOT
243  WRITE(4,202) /
244  WRITE(4,202) /
245  202 FORMAT(///, 8X, 1HE.13X, 5HTRANS, 9X, 9HREL ERROR)
246  DD 230 I = 1.35
247  WRITE(4,210) E(I), TRANS(I), RTRANS(I)
248  CONTINUE
249  WRITE(4,203) /
250  WRITE(4,203) /
251  210 FORMAT(5X, 1PE10.3, 5X, 1PE10.3, 5X, 1PE10.3, 5X, 1PE10.3, 5X)
252  CONTINUE
253  WRITE(4,204) TTOT, RRTOT
254  WRITE(4,204) /
255  WRITE(4,204) /
256  ABSORB = ABSORB/RNSP
257  WRC = WRG/RNSP
258  WR = WRL/RNSP
259  WC = WCP/RNSP
260  WC = WC / CNP
261  WRITE(4,221) ABSORB, CUTOFF
262  WRITE(4,221) /
263  221 FORMAT(///, 9HABSORB = .1PE10.3, 9HCUTOFF = .15)
264  WRITE(4,3729) NS, NR
265  WRITE(4,3728) NS, NR
266  3728 FORMAT(28H TRACKS CREATED BY SPLITTING, 18,
267  1 24H TRACKS LOST TO ROULETTE, 18)

B-7
337  WRITE(4,3729) WRL, WRL
338  3729 FORMAT(27H WEIGHT CREATED BY ROULETTE,1PE11.4,
339  1 24H WEIGHT LOST TO ROULETTE,1PE11.4),
340  WRITE(4,3730) NCO
341  3730 FORMAT(29H TRACKS LOST TO WEIGHT CUTOFF,1B)
342  WRITE(4,3731) WCP, WCO
343  3731 FORMAT(32H, WEIGHT CREATED BY, WEIGHT CUTOFF,1PE11.4,
344  1 29H WEIGHT LOST TO WEIGHT CUTOFF,1PE11.4)
345  WRITE(4,2021) TEND
346  2021 FORMAT(/,, 13HTOTAL TIME = ,1PE10.3, 6H SECONDS)
347  STOP
348  END
349
350  SUBROUTINE TRACK
351  C  CALCULATE ALL INTERSECTIONS WITH ALL THREE SURFACES
352  COMMON X, Y, Z, U, V, W, ERI, IA, NT, NP, UOLD, VOLD, WOLD, CL, CL2, CRAD2, DLS, JA
353  DLS = 1.0E10
354  IF(I.A.EQ.2) GO TO 19
355  19 DJ = 1, 3
356  300 50 D1 = -1.0
357  50 GO TO (35,160,50), J
358  GO TO 300
359  160 D1 = (CL-Y)/V
360  GO TO 280
361  280 D1 = -V/V
362  GO TO 280
363  GO TO 300
364  T1 = U**2 + W**2
365  IF(T1.EQ.0.) GO TO 300
366  A1 = (X+U + Z*W)/T1
367  B1 = (X**2 + Z**2 - CRAD2)/T1
368  T1 = A1**2 - B1
369  IF(T1.LT.0.) GO TO 300
370  T2 = SQRT(T1)
371  D1 = -A1 + T2
372  D2 = -A1 - T2
373  IF(J.A.EQ.JA) DJ = DJ = -2.*A1
374  GO TO 290
375  290 D2 = -D1
376  GO TO 290
377  IF(D1.LT.0.) GO TO 300
378  IF(D2.GT.0.) DJ = DJ
379  GO TO 300
380  300 CONTINUE
381  DLS = DLS + 1.0E-10
382  JA = JAS
383  RETURN
384  19 DJ = 2, 4
385  301 D1 = -1.0
386  51 GO TO (35,161,51,56), J
387  56 IF(V.EQ.0.) GO TO 301
388  GO TO 281
389  281 D1 = (CL-Y)/V
390  301 D1 = (CL2-Y)/V
393  GO TO 281
394   161 T1 = U**2 + W**2
395   IF(T1.EQ.0.) GO TO 301
396      A1 = (X*U + Z*W)/T1
397   398 T1 = A1**2 - B1
399   IF(T1.LT.0.) GO TO 301
400      T2 = SQRT(T1)
401      D1 = -A1 + T2
402      D2 = -A1 - T2
403   IF(J.EQ.JA) D2=D1+-2+A1
404   GO TO 291
405   291 D2 = -D1
406  IF(D1.LE.0.I GOTO301
407  IF(D2.GT.0.) D1=D2
408  IF(D1.GE.DLSS) GO TO 301
409      DSS = J
410   301 CONTINUE
411   412 DSS = DSS+1.0E-10
412   413 JA = JAS
414   RETURN
415   END
416  SUBROUTINE KLEIN(T1,T4)
418   C SAMPLE FROM KLEIN-NISHINA USING INVERSE FIT.
419   C T1=ENERGY IN, T4=ENERGY OUT, IN UNITS OF THE REST MASS
420   C OF AN ELECTRON.
421   RN=RANF(KRN)
422   423 T2=1./T1
424   T4=2.*T1+1.
425   T5=1./T4
426   T6=ALOG(T4)
427   T7=2.*T1*(1.+T1)*T5**2+4.*T2*(1.-.2.*T2*(1.+T2))*T6
428   IF(T1.LE.1.6666667) GO TO 20
429   IF(T1.LE.1.6666667) GO TO 20
430   T3=T7/T3
431   IF(RN.LE.T3) GO TO 10
432   T4=(T6-1.20397)/(1.-T3)
433   T7=3.*EXP(T4*(T3-RN))
434   435 10 T4=T7/(3.63333+2*(5.44444*T2-9.66667))
436   T7=5*T7
437   T2=RN/T3
438   T3=2.1
439   T5=1.4
440   441 20 T4=T7/(T4+T5)
442   T7=5*T7
443   T2=RN
444   T5=1.-T5
445   T3=3.*T5
446   T5=2.*T5
447   30 T7=1.*T2*(T2*(2.*T7+T4-T3+T2*(T5-T7-T4))-T7)
448   40 T4=T7*T1
SUBROUTINE ISOS

SAMPLE A DIRECTION U.V.W ISOTROPICALLY.

COMMON X,Y,Z,U,V,W,ERG,IA,HI,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)

rotate UOLD,VOLD,WOLD to U.V.W through a polar
angle whose cosine is C, and through an azimuthal
angle sampled uniformly.

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=SQRT((1.-C**2)/R)
T1=T1*R
T2=T2*R
IF(ABS(WOLD).GT.999999)GO TO 30
S=SQRT((1.-WOLD**2))
V=WOLD*C+(T1*WOLD-WOLD*T2*VOLD)/S
W=WOLD*C-T1*S
RETURN
30 U=T1
V=T2
W=WOLD*C
RETURN
END

B-10
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 GAMTEB (SEQUENTIAL VERSION) -- THE ORIGINAL

3 PROGRAM GAMTEB (OUTPUT, TAPE=OUTPUT)
4 PROGRAM GAMTEB
5
6 SCALAR MONTE CARLO CODE TO TRANSPORT .001 TO 20.0 MEV
7 GAMMA RAYS IN A CARBON CYLINDER OF LENGTH CL, RADIUS CRAD
8
9 COMMON X,Y,Z,U,V,W,ERG,WT,JA, NP
10 
11 COMMON UOLD, VOLD, WOLD, CL, CL2, CRAD2, DLS, JA
12 
13 COMMON /NEW/ KRN
14 
15 DIMENSION FIM(2), E(35)
16 
17 DIMENSION EL(35), XC(35), XPP(35), XPE(35)
18 
19 DIMENSION TRANS(35), BSCAT(35), ESCAPE(35),
20
21 DIMENSION RTRANS(35), RBSCAT(35), RESCAP(35)
22
23 DIMENSION BANK(100,8), PBL(8), IBANK(100,2), IPBL(2)
24 EQUIVALENCE (PBL,X), (IPBL, IA)
25
26 INTEGER CUTOFF
27 
28 DATA RH/2.22/
29 DATA (E(I), I=1,35) / 0.01, 0.015, 0.02, 0.03, 0.04, 0.05,
30 1 0.06, 0.07, 0.08, 0.09, 0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16,
31 2 0.17, 0.18, 0.19, 0.2, 0.21, 0.22, 0.23, 0.24, 0.25, 0.26, 0.27,
32 3 0.28, 0.29, 0.3, 0.31, 0.32, 0.33, 0.34, 0.35, 0.36, 0.37, 0.38,
39
40 DATA (XC(I), I=1,35) / 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08,
41 1 0.09, 0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17, 0.18, 0.19,
42 2 0.2, 0.21, 0.22, 0.23, 0.24, 0.25, 0.26, 0.27, 0.28, 0.29, 0.3,
43 3 0.31, 0.32, 0.33, 0.34, 0.35, 0.36, 0.37, 0.38, 0.39, 0.4, 0.41,
44
45 DATA (XPP(I), I=1,35) / 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08,
46 1 0.09, 0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17, 0.18, 0.19,
47 2 0.2, 0.21, 0.22, 0.23, 0.24, 0.25, 0.26, 0.27, 0.28, 0.29, 0.3,
48 3 0.31, 0.32, 0.33, 0.34, 0.35, 0.36, 0.37, 0.38, 0.39, 0.4, 0.41,
49
50 DATA (XPE(I), I=1,35) / 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08,
51 1 0.09, 0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17, 0.18, 0.19,
52 2 0.2, 0.21, 0.22, 0.23, 0.24, 0.25, 0.26, 0.27, 0.28, 0.29, 0.3,
53 3 0.31, 0.32, 0.33, 0.34, 0.35, 0.36, 0.37, 0.38, 0.39, 0.4, 0.41,
54
55 C CONVERT CROSS-SECTION UNITS TO BE PER CM.
56 DO 1 I=1,35
57 XC(I)=ALOG(XC(I)*RHO)
58 C IF(XPP(I).EQ.0.) XPP(I)=1.0E-12
IF(XPP(I).EQ.0.) XPP(I)=1.0E-37

IF(XPP(I).EQ.0.) XPP(I)=1.0E-123

IF(XPP(I).EQ.0.) XPP(I)=1.0E-37

XPP(I)=ALDG( XPP(I)*RHO )

XPE(I)=ALDG( XPE(I)*RHO )

EL(I)=ALDG(E(I))

1 CONTINUE

C INITIALIZE PROBLEM INPUT

NPP=500000

NPP=100

NCDL=0

WCP1=.5

WCP2=.25

NCD=0

WCD=0

WCP=0

EC = .001

NPS = 0

KRN2 = 123

CL=20.0

CRD=CL+10.

CRAD=1.0

CRAD2=CRAD**2

CUTOFF = 0

BDT=0.0

BTT=0.0

ETD=0.0

ETDT2=0.0

WRL=0.

VRO=0.

NR=0.

INBNK=0.

NBANK=0.

FIM(1)=1

FIM(1)=1.0

FIM(2)=2.0

DO 5 I = 1,35

TRANS2(I)= 0.0

BCAT2(I)= 0.0

ESCAP2(I)= 0.0

TRANS(I) = 0.0
C-5

BSCAT(I) = 0.0

5 ESCAPE(I) = 0.0

C START A HISTORY

CALL SECOND(1)

10 NPS = NPS + 1

IF(NPS .EQ. NPP+1) GO TO 140

C SET SOURCE VALUES

ERG = 6.0

WT = 1.0

U = 0.0

V = 1.0

W = 0.0

X = 0.0

Y = 0.00001

Z = 0.0

JA = 0

IA = 1

C GENERATE NEW RANDOM SEED

XJUNK = RAND0(KRN2)

KRN = KRN2

C CALCULATE DISTANCE DLS TO NEXT SURFACE INTERSECTION

FOR ALL THREE SURFACES AND ALSO THE NUMBER JA OF THE NEXT SURFACE INTERSECTED

C 20 JA=0

CALL TRACK

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 INTERPOLATION TO GET CROSS SECTIONS AS F(ERG)

F = (-ALOC(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 CALCULATE DISTANCE TO NEXT COLLISION

S = -ALOC(RANF(KRN))/XST

C SEE IF COLLISION IS STILL INSIDE CYLINDER

IF(S .LT. DLS) GO TO 60

IF(S .LT. L.5) JA = 0

DLO = 30.50.53.52

GO TO (DLO, 30, 31, 32) JA

BSCAT(I) = BSCAT(I) + WT

BTOY = BTOY + WT
GO TO 31
52 TRANS(I) = TRANS(I) + WT
TTOP = TTOP + WT
GO TO 11
50 ESCAPI(I) = ESCAPI(I) + WT
ETOT = ETOT + WT
GO TO 11
C CROSS INTERNAL SURFACE SPLIT OR ROULETTE
53 IAP = IA
IA = 2 - IA
T1 = FIM(IA)/FIM(IAP)
IF(T1 . LT. 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 = 1.
WT = WT/T1
NS = NS + NP
NBANK = NBANK + NP
INBNK = INBNK + 1
DO 59 IX = 1, 8
59 BANK(INBNK, IX) = PBL(IX)
DO 61 IX = 1, 2
61 IBANK(INBNK, IX) = IPBL(IX)
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
IF(IBANK(INBNK, 2) . EQ. 0) INBNK = INBNK - 1
GO TO 20
C COLLISIONS
60 JA = 0
X = X + U8
Y = Y + V8
Z = Z + H8
NCOL = NCOL + 1
C SURVIVAL BIAS
WTSAV = WT
WT = WT * (1. - XSPE/XST)
ABSORB = ABSORB + (WTSAV - WT)
XST = XST * XSPE
C WEIGHT CUTOFF
223 IF(WT, GT, WCP2) GO TO B32
226 IF(WT+FIM(IA), LT, RANF(KRN)*WCP1+FIM(1)) GO TO 642
227 WTSAV=WT
228 WT=WCP1+FIM(1)/FIM(IA)
229 WCP=WCP+(WT-WTSAV)
230 B32 CONTINUE
231 IF(RANF(KRN), GE, XSC/XSTS) QQ_TO_100
232 T1 = 1.956917*ERG
233 T GET NEW ENERGY T4 AND COMPTON SCATTERING ANGLE
234 CALL KLEIN(T1, T4)
235 CSA = 1. +1./T1-1./T4
236 T5 = 5.11008*T4
237 IF(ABS(CSA), GT, 1.) CSA=SIGN(1., CSA)
238 ERG = T5
239 C SEE IF NEW ENERGY IS LESS THAN CUTOFF
240 IF(ERG, GT, EC) GO TO 70
241 CUTOFF = CUTOFF + 1
242 GO TO 11
243 C MAKE COMPTON ANGLE RELATIVE TO PROBLEM COORDINATE SYSTEM
244 CALL ROTAS(CSA)
245 CALL KLEIN(T1, T4)
246 CALL KLEIN(T1, T4)
247 WOLD = W
248 VOLD = V
249 GO TO 20
250 C PAIR PRODUCTION
251 100 ERG = 0 511008
252 WT = 2.*WT
253 C CHECK ENERGY CUTOFF
254 IF(ERG, GT, EC) GO TO 110
255 CUTOFF = CUTOFF + 1
256 GO TO 11
257 C ISOTROPIC EMISSION IN LAB SYSTEM
258 CALL ISOS
259 110 GO TO 20
260 C PHOTOELECTRIC ABSORPTION
261 130 ABSORB = ABSORB + WT
262 C NOW HANDLED BY SURVIVAL BIASING
263 C TERMINATE PARTICLE TO WEIGHT CUTOFF
264 642 NCO=WCO+WT
265 NCO=NCO+1
266 GO TO 11
267 234 DD B29 1=1.35
268 BSCAT(I)=BSCAT(I)+BSCAT(I)
269 BSCAT2(I)=BSCAT2(I)+BSCAT2(I)**2
270 TRANS(I)=TRANS(I)+TRANS(I)
271 TRANS2(I)=TRANS2(I)+TRANS2(I)**2
272 ESCAPE(I)=ESCAPE(I)+ESCAPE(I)
273 ESCAPE2(I)=ESCAPE2(I)+ESCAPE2(I)**2
274 ESCAPE(I)=ESCABI(I)
275 ESCAPE2(I)=ESCABI2(I)
276 BSCAT(I)=0
277 BSCAT2(I)=0
278 TRANS(I)=0
279 TRANS2(I)=0
280}
C

C PRINT OUTPUT
140 NPS = NPS - 1
C
CALL SECOND(T1)
TEND = T1 - TO
WRITE(4,7634) NCOL
7634 FORMAT(5HNCOL,110)
WRITE(4,1401)
WRITE(4,150) NPS
150 FORMAT(6HNCOL = ,16)
WRITE(4,200)
200 FORMAT(//,8X,1HE,13X,5HSCAT,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
RESCAP(I) = 0.0
GO TO 209
207 RESCAP(I) = SQRT((ESCAPE2(I)-ESCAPE(I)**2)/RNPS)
RESCAP(I) = RESCAP(I)/ESCAPE(I)
209 WRITE(4,210) E(I),BSCAT(I),RBScat(I)
220 CONTINUE
TDT = TDT/RNPS
TTDT2 = TTDT/RNPS
BTDT = BTDT/RNPS
BTTDT2 = BTDT2/RNPS
ETOT = ETOT/RNPS
229 CONTINUE
BTDT = BTDT+BTDT
TTDT = TTDT+TTDT
ETOT = ETOT+ETOT
BTTDT2 = BTTDT2+BTDT**2
ETOT2 = ETOT2+ETOT**2
BTTDT2 = BTTDT2+BTDT**2
ETOT2 = ETOT2+ETOT**2
ETOT = ETOT**2
TTOT = TTOT+TTDT
ETOT = ETOT+ETOT
BTDT = BTDT+BTDT
TTDT = TTDT+TTDT
ETOT = ETOT+ETOT
BTTDT2 = BTTDT2+BTDT**2
ETOT2 = ETOT2+ETOT**2
ETOT2 = ETOT2+ETOT**2
```fortran
337  IF (TTOT NE 0.0) GO TO 2000
338      RTTOT = 0.0
339      GO TO 2001
340 2000 RRTOT = SQRT((TTOT2 - TTOT**2)/RNPS)
341      RTTOT = RRTOT/TTOT
342 2001 IF (ETOT NE 0.0) GO TO 2002
343      RTTOT = 0.0
344      GO TO 2003
345 2002 RBTOT = SQRT((BTOT2 - BTOT**2)/RNPS)
346      RTTOT = RBTOT/BTOT
347 2003 IF (ETOT NE 0.0) GO TO 2004
348      RRTOT = 0.0
349      GO TO 2005
350 2004 RETOT = SQRT((ETOT2 - ETOT**2)/RNPS)
351      RBTOT = RETOT/ETOT
352 2005 CONTINUE
353      WRITE(4,2020) BTOT, RBTOT
354 2020 FORMAT (/6X,5HTOTAL,9X,1PE10.5X,OPF7.4)
355      WRITE(4,201)
356 201 FORMAT (/8X,1HE.13X,6HESCAPE,8X,9HREL ERROR)
357      DO 225 I=1,35
358      WRITE(4,210) E(I),ESCAPE(I),RESCAP(I)
359 225 CONTINUE
360      WRITE(4,202) ETOT, RETOT
361      WRITE(4,202)
362 202 FORMAT (/8X,1HE.13X,5HTRANS,9X,9HREL ERROR)
363      DO 230 I=1,35
364      WRITE(4,210) E(I),TRANS(I),RTRANS(I)
365 210 FORMAT(5X,1PE10.3X,1PE10.3X,OPF6.3)
366 230 CONTINUE
367      WRITE(4,2020) TTOT, RRTOT
368      ABSORB = ABSORB/NPS
369      WRC=WRC/NPP
370      WRL=WRL/NPP
371      WCP=WCP/NPP
372      WCD=WCD/NPP
373      WRITE(4,221) ABSORB, CUTOFF
374 221 FORMAT (/8X,9HABSORB = ,1PE10.3X,9HCUTOFF = ,15)
375      WRITE(4,3729) NS, NR
376 3729 FORMAT(28H TRACKS CREATED BY SPLITTING,1B)
377      1 24H TRACKS LOST TO ROULETTE,1B)
378      WRITE(4,3729) WRC,WRL
379 3729 FORMAT(28H WEIGHT CREATED BY ROULETTE,1PE11.4)
380      1 24H WEIGHT LOST TO ROULETTE,1PE11.4)
381      WRITE(4,3730) NCD
382 3730 FORMAT(29H TRACKS LOST TO WEIGHT CUTOFF,1B)
383      WRITE(4,3731) WCP,WCD
384 3731 FORMAT(32H WEIGHT CREATED BY WEIGHT CUTOFF,1PE11.4)
385      1 29H WEIGHT LOST TO WEIGHT CUTOFF,1PE11.4)
386      WRITE(4,3731) TEND
387 2021 FORMAT (/13HTOTAL TIME = ,1PE10.3, BH SECONDS)
388      STOP
389      END
390
391
392
```

SUBROUTINE TRACK

CALCULATE ALL INTERSECTIONS WITH ALL THREE SURFACES

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

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

COMMON /NEW/ KRN

DLSS = 1.0E10
IF(IA.EQ.2) GO TO 19
DO 300 J=1,3
D1 = -1.0
GO TO (55, 160, 50), J

ENDIF

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
IF(J.EQ.JA) THEN
D1=-2.*A1
D2=-2.*A1
END IF
GO TO 290

280 D2 = -D1
290 IF(D1.LT.0.) GO TO 300
IF(D2.GT.0.) D1=D2

300 IF(D1 .GE. DLSS) GO TO 300
JAS = J
300 CONTINUE
DLSS = D1
JAS = JAS

19 DO 301 J=2,4
D1 = -1.0
GO TO (56, 161, 51, 56), J
IF(V.EQ.0.) GO TO 301
D1 = (CL-Y)/V
GO TO 281

56 IF(V.EQ.0.) GO TO 301
449        D1 = (CL2-Y)/V
450        GO TO 281
451        161 T1 = U**2 + M**2
452        IF(T1.LE.0.) GO TO 301
453        A1 = (X*U + Z*W)/T1
454        B1 = (X**2 + Z**2 - CRAD2)/T1
455        T1 = A1**2 - B1
456        IF(T1.LT.0.) GO TO 301
457        T2 = SQRT(T1)
458        D1 = -A1 + T2
459        D2 = -A1 - T2
460        C IF (J.EQ.JA) D2=D1=-2.*A1 NEW
461        IF (J.EQ.JA) THEN
462          D1=-2.*A1 NEW
463          D2=-2.*A1 NEW
464        END IF
465        GO TO 291
466        281 D2 = -D1
467        291 IF(D1.LE.0.) GO TO 301
468        IF(D2.GT.0.) D1=D2
469        IF(D1.GE.DLSS) GO TO 301
470        JAS = J
471        DLSS = D1
472        301 CONTINUE
473        DL5 = DLSS+1.QE-10
474        JA = JAS
475        RETURN
476        END
477
478 SUBROUTINE KLEIN(T1,T4)
479 SAMPLE FROM KLEIN-NISHINA USING INVERSE FIT
480 T1=ENERGY IN. T4=ENERGY OUT. IN UNITS OF THE REST MASS
481        C OF AN ELECTRON.  
482        C
483        COMMON /NEW/ KRN
484        d13 BUGFIX
485        RN=RANF(KRN)
486        T2=1./T1
487        T4=2.*T1+1.
488        T5=1./T4
489        T6=ALOG(T4)
490        T3=2.*T1+(1.-T1)+T3**2+4.*T2+1.-2.*T2*(1.+T2)))*T6
491        IF(T1.LE.1.6666667)GO TO 20
492        T7=1.63898+T2*(1.62937+T2-1.00796)
493        T3=T7/T3
494        IF(RN.LE.T3)GO TO 10
495        T4=(T6-1.20397)/(1.-T3)
496        T7=3.*EXP(T4*(T3-RN))
497        GO TO 40
498        10 T4=T7/(3.63333+T2*(5.44444+T2-4.66667))
499        T7= 5*T7
500        T2=RN/T3
501        T3=2.1
502        T5=1.4
503        GO TO 30
504        20 T4=T3/(T4+T5)

C-11
C-12
IF(ABS(WOLD).GT.999999)GO TO 30

S=SQRT(1.-WOLD**2)

U=WOLD*C+(T1*UOLD+WOLD*WOLD-T2*VOLD)/S

V=WOLD*C+(T1*VOLD+WOLD*T2*VOLD)/S

W=WOLD*C-T1*S
RETURN

U=1.
V=T2
W=WOLD*C
RETURN

END

SUBROUTINE SECOND. (T)

T=0
RETURN
END

REAL FUNCTION RANDCRN)

KERN = MOD(1+7421*KERN, 131072)
RANDCRN = FLOAT(KERN)/131072.
RETURN

END

REAL FUNCTION RANF(KERN)

KERN = MOD(1+9621*KERN, 131072)
RANF = FLOAT(KERN)/131072.
RETURN

END
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
LGDF Project File:

project_title_(Scalar Monte Carlo Transport Code)
program_name_(GAMTEB)
machine_(VAX)
language_(FORTRAN)
wirelist_(Wirelist)
data_directory_(.)
noptrace_(.)
nogtrace_(.)
set up problem constants p10

run particle histories and print report p01

- problem constants d03
- converted cross section tables d04
- source values d05

- prompt for # of particles d09
- done d15
- GAMTEB report file d07

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

Apr 17 14:01 1985  Wirelist Page 1

*** DEFINE DATA PATHS

<table>
<thead>
<tr>
<th>tag</th>
<th>short name</th>
<th>descriptive name</th>
</tr>
</thead>
<tbody>
<tr>
<td>_defdp(d01, &quot;user input&quot;, &quot;No. of Particles to run&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_defdp(d02, &quot;NPP&quot;, &quot;No. of Particles to run&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_defdp(d03, &quot;constants&quot;, &quot;problem constants&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_defdp(d04, &quot;cs tables&quot;, &quot;cross section tables&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_defdp(d05, &quot;src values&quot;, &quot;source values&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_defdp(d06, &quot;run stats&quot;, &quot;run statistics&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_defdp(d07, &quot;report&quot;, &quot;GAMTEB report file&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_defdp(d09, &quot;prompts&quot;, &quot;prompt for no. of particles&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_defdp(d10, &quot;NPPR&quot;, &quot;particle count for report&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_defdp(d11, &quot;NPPC;NPC&quot;, &quot;particle completion control&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_defdp(d12, &quot;PSC&quot; &quot;particle start control&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_defdp(d13, &quot;KERN&quot;, &quot;random seed&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_defdp(d14, &quot;signal&quot;, &quot;particle history completion signal&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_defdp(d15, &quot;done&quot;, &quot;run completion interlock&quot;)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*** DEFINE PROGRAMS & NETWORKS

<table>
<thead>
<tr>
<th>tag</th>
<th>short name</th>
<th>descriptive name</th>
</tr>
</thead>
<tbody>
<tr>
<td>_defpn(p00, &quot;GAMTEB&quot;, &quot;Scalar Monte Carlo Transport Code&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_defpn(p01, &quot;RUNPR&quot;, &quot;run particle histories and print report&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_defpn(p02, &quot;RUNPH&quot;, &quot;run particle history&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_defpn(p10, &quot;SETUP&quot;, &quot;set up problem constants&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_defpn(p12, &quot;PARAM&quot;, &quot;init GAMTEB run parameters&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_defpn(p13, &quot;PCOUNT&quot;, &quot;setup for particle counting&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_defpn(p14, &quot;REPORT&quot;, &quot;format GAMTEB report&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_defpn(p16, &quot;GENRAN&quot;, &quot;generate random seed for next particle&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_defpn(p17, &quot;RNHIST&quot;, &quot;run history for one particle and offspring&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_defpn(p18, &quot;CKCOMP&quot;, &quot;check for run completion&quot;)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*** 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)

  _endsprog(p10, [p00])
  _sys(p01, [p00])
    _in(d03, NC)
Apr 17 14:01 1985 Wirelist Page 2

```
-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, RO)
  _in(d11, CL)
  _in(d14, CL)
  _shared(d06, d06a, NC, RO)
  _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
  GIERS - 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
  IGCUF - 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
  GIRNS2(35) - sum of squares of GTRANS
  GESCP2(35) - sum of squares of GESCAP
  GBST2(35) - sum of squares of GBSCAT
  GIRNSI(35) -
  GBSTI(35) -
  GESPI(35) -
  GTOTI - back-scattered weight total
GETOTI - escaped weight total
GETOTI - transmitted weight total
GETOTI - sum of squares of BTOTI
GETOTI - sum of squares of TTOTI
GETOTI - sum of squares of ETOTI

DO7
GETOTI - escaped weight total
GETOTI - transmitted weight total
GETOTI - sum of squares of BTOTI
GETOTI - sum of squares of TTOTI
GETOTI - sum of squares of ETOTI

DO8
[not assigned]

DO9
prompt for # of particles

DO10
NFPR - particle count for report

DO11 - particle completion control
NFPC - no. of particle history summary statistics (d14) expected
   (= NPP (d02))
NPC - particle history completion counter

DO12 - particle start control
NPPS - no. of particles to start
   (= NPP (d02))
NPS - particle history start counter

DO13
KERN - first random number seed ($KRN)

DO14
NCOL - number of collisions
NCO - tracks lost to weight cutoff
WCO - weight lost to weight cutoff
WCP - weight created by weight cutoff
WRP - weight lost in russian roulette
NR - tracks lost to roulette
WRC - 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 -
ESCAPPI -
BTOTI - back-scattered weight total
ETOTI - escaped weight total
TTOTI - transmitted weight total
BTOT2 -
TTOT2 -
ETOT2 -
INTEGER ID01

INTEGER NPP

REAL EC, FIM (2)

REAL E, RHO, CL, CL2, CRAD, CRAD2, WCP1, WCP2, EC, FIM

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

REAL EL (35), XC (35), XPP (35), XPE (35)
::: :::
d05
::: :::
GIERS, GIWT, GIU, GIV, GIW, GIX, GIY, GIZ, IGIA
REAL GIERS, GIWT, GIU, GIV, GIW, GIX, GIY, GIZ
INTEGER IGIA

::: :::
d06
::: :::
IGNCOL, IGNCO, GWCO, GWCP, GWRL, IGNR, GWRG, IGCUIF,
+ GABSOR, IGNS,
+ TRANS, ESCAPE, BSCAT, TRANS2, ESCAP2, BSCAT2,
+ BTOT2, TTOT2, ETOT2, ETOT, BTOT, TTOT

INTEGER IGNCOL, IGNCO, IGNR, IGCUIF, 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

D-13
CL, CL2, CRAD, CRAD2, WCP1, WCP2, EC
REAL CL, CL2, CRAD, CRAD2, WCP1, WCP2, EC

INTEGER NPPR

INTEGER NPC, NPPC

INTEGER NPPS, NPS, KRN2
KERN

INTEGER KERN

INTEGER ID15

INTEGER ID015
program _ ( pl0, fp001).
begin

CL = 2.0
CL2 = CL + 10.
CRAD = 1.0
CRAD2 = CRAD**2
WCP1 = 5.
WCP2 = 23.
EC = .001
FIM(1) = 1.0
FIM(2) = 2.0
set_(003)

C CONVERT CROSS-SECTION UNITS TO BE PER CM.

DO 1 1 = 1, 35
1 XC(I) = ALDG( XC(I) * RHOD )
IF(XPP(I).EQ.0.) XPP(I) = 1.0E-37
IF(XPE(I).EQ.0.) XPE(I) = 1.0E-37
XPP(I) = ALDG( XPP(I) * RHOD )
XPE(I) = ALDG( XPE(I) * RHOD )
EL(I) = ALDG(E(I))
1 CONTINUE
set_(004)

C SET SOURCE VALUES

GIERQ = 6.0
GIWT = 1.0
GIV = 1.0
GIW = 0.0
GIX = 0.0
GIY = .0000001
GIZ = 0.0
IIGA = 1
set_(005)
suspend_
end_(p10)

_d_(d03)
BLOCK DATA
_d_(d04)

DATA RHD/2.22/
DATA (X(I),I=1,35)/ .001, .0015, .002, .003, .004, .005,
1 .006, .008, .01, .013, .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)/ .015, .0296, .0451, .0717, .0913
1 .102, .119, .128, .137, .152
2 .160, .165, .169, .172, .169
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, .0, .0, .0, .0, .0, .0, .0, .0, .0, .0, .0, .0, .0, .0, .0, .0, .0, .0
2 .000316, .000923, .00153, .00208, .00256, .00343.
3 .00414, .00547, .00652/

D-17
<table>
<thead>
<tr>
<th></th>
<th>DATA (XPE(1), I=1,35)</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>57</td>
<td></td>
<td>2010, 432, 280, 87, 7, 37.3, 16.9,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>1</td>
<td>10.4, 4.01, 1.91, 4.89, 1.92, 0.0491, 0.0186, 0.00887,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>2</td>
<td>0.00481, 0.00179, 0.000862, 0.000234, 0.0000918,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>4</td>
<td>0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>END</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
program(p12,p01)
begin
   INITIALIZE PROBLEM INPUT
   C
   lockio
   WRITE(*,*) 'p12: ENTER NO. OF PARTICLES TO RUN'
   NPP=500000
   READ(5,*) NPP
   unlockio
   IF (NPP.EQ.0) THEN
      set(d15)
      suspend
   ELSE
      set(d02)
   ENDIF
   C
   IGNCOL=0
   IGNCO=0
   GWCO=0
   GWC=0
   GWL=0.
   GWG=0.
   GABS=0.
   IGNR=0.
   IG=0
   IGUT= 0
   IGNS=0
   DD 5 I = 1.35
   TRANS(I)= 0.0
   BSCAT(I)= 0.0
   ESCAP2(I)= 0.0
   TRANS(I).= 0.0
   BSCAT(I)= 0.0
   ESCAPE(I)= 0.0
   5 CONTINUE
   BTOT=0.0
   BTOT2=0.0
   ITOT=0.0
   TTOT2=0.0
   ETOT=0.0
   ETOT2=0.0
   set(d06)
   suspend
end(p12)
program (p13[p01])
begin
C
KRN = 123454321
C -- CHANGE TO KRN2 DUE TO ADDITION OF SECOND R.N. GENERATOR
C -- SET UP PARTICLE START CONTROL
C
KRN2 = 123
NPPS=NPP
NPC=0
set_(d12)
C -- SET UP PARTICLE COMPLETION CONTROL
NPC=NPP
NPPC=NPP
set_(d11)
C -- MAKE COPY OF NPP FOR REPORT PROCESS
NPPR=NPP
set_(d10)
clear_(d02)
suspend
end_(p13)
program (p14,p01))
REAL RTRANS(35),RSCAT(35),RESCAP(35)

begin

C PRINT OUTPUT

C -- NPP AND NPS ARE USED INTERCHANGEABLY IN THE ORIGINAL CODE

C 140 NPS = NPS - 1
C 140 NPS=NPPR
C -- FIX UP NAME ALIAS FOR NPP
NPP=NPPR
C -- GET EXCLUSIVE ACCESS TO I/O FOR PRINT FILE
LOCKIO

WRITE(*,7634) IONCOL
17634 FORMAT(5H NCOL, I10)
WRITE(4, 1401)
1401 FORMAT(7HSCLERT, /)
WRITE(4, 150) NPS
150 FORMAT(5H NPS = , I6)
WRITE(4, 200)

DO 220 I=1,35
      RNPS=NPS
      TRANSCI)=TRANS(1)/RNPS
      BSCAT(I)=BSCAT(I)/RNPS
      ESCAPE(I)=ESCAPE(I)/RNPS
      TRANS2(I)=TRANS2(I)/RNPS
      BSCAT2(I)=BSCAT2(I)/RNPS
      ESCAP2(I)=ESCA P2(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)
204 IF(BSCAT(I).NE.0.0)GO TO 209
209 WRITE(4,210) E(I),BSCAT(I),RBSCAT(I)
210 CONTINUE

TTOT = TTOT/RNPS
220 CONTINUE

ETOT = ETOT/RNPS
ETOT2 = ETOT2/RNPS
IF(TTOT.NE.0.0) GO TO 2000
2000 RTTOT = SQRT((TTOT2 - TTOT**2)/RNPS)
57  RTOT = RTIQ/TTOT
58  2001 IF(BTOT .NE. 0.0) GO TO 2002
59  RBOT = 0.0
60  GO TO 2003
61  2002 RBOT = SORT((BTOT2 - BTOT*2)/RNPS)
62  RTOT = RBOT/BTOT
63  2003 IF(ETOT.NE.0.0) GO TO 2004
64  RETOT = 0.0
65  GO TO 2005
66  2004 RETOT = SORT((ETOT2 - ETOT**2)/RNPS)
67  RETOT = RETOT/ETOT
68  2005 CONTINUE
69  WRITE(4,2020) BTOT, RBOT.
70  2020 FORMAT(/,6X,5HTOTAL,9X,1PE10.3,5X,0PF7.4)
71  WRITE(4,201)
72  201 FORMAT(/,8X,1HE,13X,6HESCAPE,8X,9HREL ERROR)
73  DO 225 I=1,1.35
74  WRITE(4,210) E(I),ESCAPE(I),RESCAP(I)
75  225 CONTINUE
76  WRITE(4,2020) ETOT, RETOT
77  WRITE(4,202)
78  202 FORMAT(/,8X,1HE,13X,5HTRANS,9X,9HREL ERROR)
79  DO 230 I=1,1.35
80  WRITE(4,210) E(I),TRANS(I),RTRANS(I)
81  210 FORMAT(5X,1PE10.3,5X,1PE10.3,5X,0PF6.3)
82  230 CONTINUE
83  WRITE(4,2020) TTOT, RTTOT
84  GABSOR = GABSOR/NPS
85  GWRG=GWRG/NPP
86  GWRL=GWRL/NPP
87  GWCP=GWCP/NPP
88  GWC0=GWC0/NPP
89  WRITE(4,221) GABSOR, IG0CUTF
90  221 FORMAT(/,9HABSOR8 = 1PE10.3,5X,9HCUTOFF = ,15)
91  WRITE(4,3728) IGN8, IGNR
92  3728 FORMAT(26H TRACKS CREATED BY SPLITTING,18,
93  1 24H TRACKS LOST TO ROULETTE,18)
94  WRITE(4,3729) GW0, GWRG
95  3729 FORMAT(27H WEIGHT CREATED BY ROULETTE,1PE11.4,
96  1 24H WEIGHT LOST TO ROULETTE,1PE11.4)
97  WRITE(4,3730) IGN0C
98  3730 FORMAT(29H TRACKS LOST TO WEIGHT CUTOFF,18)
99  WRITE(4,3731) GWCP,GWCO
100  3731 FORMAT(32H WEIGHT CREATED BY WEIGHT CUTOFF,1PE11.4,
101  1 29H WEIGHT LOST TO WEIGHT CUTOFF,1PE11.4)
102  WRITE(4,2021) TEND
103  2021 FORMAT(/,9HTOTAL TIME = 1PE10.3,8H SECONDS)
104  C -- RETURN ACCESS TO I/O
105  unlock
106  clear_(d06)
107  clear_(d10)
108  suspend
109  end_(p14)
program (p16,p021)
begin
10 NPS = NPS + 1
IF(NPS GT NPPS) THEN
GO TO 140

C clear (d12)

ELSE
GENERATE NEW RANDOM SEED
XJUNK = RANDO(KRN2)
KERN = KRN2
set (d13)
ENDIF

suspend

END
program (p17,p021)

SCALAR MONTE CARLO CODE TO TRANSPORT .001 TO 20.0 MEV GAMMA RAYS IN A CARBON CYLINDER OF LENGTH CLR, RADIUS CRAD

REAL BANK(100,2), IPBL(2)        BUGFIX
INTEGER IBANK(100,2), IPBL(2)     BUGFIX
EQUIVALENCE (PBL(1), X), (PBL(2), Y), (PBL(3), Z), (PBL(4), U), (PBL(5), V), (PBL(6), W) BUGFIX
+ (PBL(7), ERG), (PBL(8), WT), (IPBL(1), IA), (IPBL(2), NP) BUGFIX
+ U=GIU, V=GIV, W=GIW, X=GIX, Y=GIY, Z=GIZ, IA=GI1A BUGFIX

INTEGER INBNK, NBANK ALG CH
INTEGER KRN
INTEGER CUTOFF

begin
read (d13)

C -- MAKE LOCAL COPY OF RANDOM SEED
KRN=KERN

C -- MAKE LOCAL COPIES OF INITIAL SOURCE VALUES
ERG=GIERG
WT=GIWT
V=GIV
W=GIW
X=GIX
Y=GIY
Z=GIZ
IA=GI1A

C
NCO=0
NC=0
WCO=0
WCP=0
WRL=0.
WR=0.
NR=0
CUTOFF= 0
ABSORB=0
NS=0

DQ 5 I = 1.35

BSCAT(I)=0.
TRANS(I)=0.
ESCAPI(I)=0.

5 CONTINUE

BTOTI=0.
ITOTI=0.

C -- INITIALIZE LOCAL PARTICLE BANK INDEXES
INBNK=0

C

56 C

D-24
57 C Calculate distance DLS to next surface intersection
58 C for all three surfaces and also the number JA of the
59 C next surface intersected
60 C
61 20 JA=0
62 CALL TRACK (IA, JA, X, Y, Z, U, V, W, CL, CL2, CRAD2, DLS)
63 C
64 C Find energy pointer for cross sections and tallys
65 C
66 DO 30 IE = 1, 35
67 IF (E (IE) .GT. E (IE)) Go to 30
68 I = IE
69 Go to 31
70 30 CONTINUE
71 C
72 C Interpolation to get cross sections as F (ERG)
73 C
74 31 F = (ELO (ERG) - EL (I - 1)) / (EL (I) - EL (I - 1))
75 XSC = EXP (X (I - 1) + F * (X (I) - X (I - 1)))
76 XPE = EXP (XPE (I - 1) + F * (XPE (I) - XPE (I - 1)))
77 XST = XSC + XSPP + XSPE
78 C Calculate distance to next collision
79 C
80 S = -ALGO (RANF (KRN)) / XST
81 C
82 C See if collision is still inside cylinder
83 C If not, do tallys; if so, do collision physics
84 C
85 IF (E (S.LT. DLS)) Go to 60
86 X = X + U * DLS
87 Y = Y + V * DLS
88 Z = Z + W * DLS
89 Go to (42, 30, 32, 35, 52) JA
90 42 BSCATI (I) = BSCATI (I) + WT
91 BTOTI = BTOTI + WT
92 Go to 11
93 52 TRANS I (I) = TRANS I (I) + WT
94 TTOTI = TTOTI + WT
95 Go to 11
96 50 ESCAPI (I) = ESCAPI (I) + WT
97 ETOTI = ETOTI + WT
98 Go to 11
99 C Cross internal surface, split or roulette
100 53 IAP = I A
101 I A = 2 - I A / 2
102 T1 = FIM (I A) / FIM (I A)
103 IF (T1 .GT. 1.0) Go to 57
104 C Russian roulette
105 IF (I A .LT. RANF (KRN)) Go to 58
106 WTSAV = WT
107 WT = WT / T1
108 WRG = WRG + (WT - WTSAV)
109 Go to 20
110 C Killed in Russian roulette
111 58 WRL = WRL + WT
112 NR = NR + 1

D-25
Apr 30 13:39 1985 p17.m Page 3

113  GO TO 11
114  C SPLITTING
115   57 NP=TI-1.
116   WT=WT/T1
117   NS=NS+NP
118   NBANK=NBANK+NP
119   INBNK=INBNK+1

120   DO 59 IX=1,B
121   59 BANK(INBNK,IX)=PBL(IX)
122   DO 61 IX=1,2
123   61 IBANK(INBNK,IX)=IPBL(IX)

125  C CHECK BANK BEFORE STARTING NEW PARTICLE
126   11 IF(NBANK.EQ.0) GO TO 234
127   DO 521 IX=1,B
128   521 PBL(IX)=BANK(INBNK,IX)
129   DO 522 IX=1,1
130   522 IPBL(IX)=IBANK(INBNK,IX)
131   NBANK=NBANK+1
132   IF(IBANK(INBNK,2).EQ.0) INBNK=INBNK-1
133   GO TO 20

135  C COLLISIONS
136   60 JA = 0
137   Y=Y+V*S
138   Z=Z+U*S
139   NCOL=NCOL+1

142  C SURVIVAL BIAS
143   WTSAV=WT
144   WT=WT*(1.-XSPE/XST)
145   ABSORB=ABSORB+(WTSAV-WT)
146   XSTSB=XST-XSPE
147  C WEIGHT CUTOFF
148   IF(WT.GT.WCP2) GO TO 832
149   IF(WT*FIM(IA).LT.RANF(KRN)*WCP1*FIM(1)) GO TO 642
150   WTSAV=WT
151   WCP=WCP+FIM(1)/FIM(IA)
152   WCP=WCP+(WT-WTSAV)
153   832 CONTINUE
154   IF(RANF(KRN).GE.XSC/XSTSB) GO TO 100
155   T1 = 1.956917*ERG
156  C GET NEW ENERGY T4 AND COMPTON SCATTERING ANGLE
157   CALL KLEIN(T1,T4,KRN) CALL KLEIN(T1,T4,KRN)
158   CSA = 1.+/T4-1./T4
159   T5 = .51008*T4
160   IF(T5(CSA,T4).GT.1.) CSA=SIGN(1.,CSA)
161   ERG = T5

162  C SEE IF NEW ENERGY IS LESS THAN CUTOFF
163   IF(ERG.GT.EC) GO TO 70
164   CUTOFF = CUTOFF + 1
165   GO TO 11
166  C MAKE COMPTON ANGLE RELATIVE TO PROBLEM COORDINATE SYSTEM
168   70 UOLD = U

D-26
167  VOLD = V
170  WOLD = W
171  CALL ROTAS(CSA, KRN, U, V, W, VOLD, VOLD, WOLD)  ARGs
172  GO TO 20
173  C  PAIR PRODUCTION
175  100. ERG = 0.51100B
176  WT = 2. *WT
177  C  CHECK ENERGY CUTOFF
179  IF (ERG.GT.EC) GO TO 110
180  CUTOFF = CUTOFF + 1
181  GO TO 11
182  C  ISOTROPIC EMISSION IN LAB SYSTEM
184  110 CALL ISOS(U, V, W, KRN)  ARGs
185  GO TO 20
186  C  PHOTOELECTRIC ABSORPTION
187  C NOW HANDLED BY SURVIVAL BIASING
189  C 130 ABSORB = ABSORB + WT
190  C  GO TO 11
191  C  TERMINATE PARTICLE TO WEIGHT CUTOFF
192  642 WCO=WCO+WT
193  NCO=NCO+1
194  GO TO 11
195  C  GET EXCLUSIVE UPDATE ACCESS TO RUN STATISTICS
196  234 AREAD(D06)
197  IGNCO=IGNCO+NCO
198  IGNCO=IGNCO+NCO
199  GWCO=GWCO+WC0
200  GWCP=GWCP+WC0
201  GWRL=GWRL+WRL
202  GWRC=GWRC+WRG
203  IGN=IGN+NR
204  IGUTF=IGUTF+CUTOFF
205  GABSOR=GABSOR+ABSORB
206  IGNS=IGNS+NS
207  DO 829 I=1,35
208  BSCAT(I)=BSCAT(I)+BSCATI(I)
209  BSCAT2(I)=BSCAT2(I)+BSCATI(I)**2
210  TRANS(I)=TRANS(I)+TRANSI(I)
211  TRANS2(I)=TRANS2(I)+TRANSI(I)**2
212  ESCAP2(I)=ESCAPE2(I)+ESCAPEI(I)**2
213  ESCAP2(I)=ESCAPE2(I)+ESCAPEI(I)**2
214  829 CONTINUE
215  BTOT=BTOT+BTOTI
216  TOT=TOT+TOTI
217  EOT=EOT+ETOTI
218  BTOT2=BTOT2+BTOTI**2
219  TOT2=TOT2+TOTI**2
220  EOT2=EOT2+ETOTI**2
221  C  GIVE BACK UPDATE ACCESS TO RUN STATISTICS
222  UNREAD(D06)
223  C  SIGN OF COMPLETION OF THIS PARTICLE HISTORY
224  SET(D14)
SUBROUTINE TRACK(IA, JA, X, Y, Z, U, V, W, CL, CRAD2, DLS)

CALCULATE ALL INTERSECTIONS WITH ALL THREE SURFACES

DLSS = 1D-10

GO TO (55, 160, 50), J

IF(V .EQ. 0.) GO TO 300

D1 = (CL-Y)/V

GO TO 280

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

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

JA = J

DLSS = D1

300 CONTINUE

DLS = DLSS + 1D-10

JA = JAS RETURN

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 = (CL-Z)/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
SUBROUTINE KLEIN(T1, T4, KRN)

SAMPLE FROM KLEIN-NISHINA USING INVERSE FIT.

T1 = ENERGY IN, T4 = ENERGY OUT, IN UNITS OF THE REST MASS OF AN ELECTRON.

RN = RANF(KRN)
T2 = 1. / T1
T4 = 2. * T1 + 1.
T5 = 1. / T4
T6 = ALOG(T4)

IF(T1 .LE. 1.6666667) GO TO 20
T7 = 1.69898 + T2 * (62537 * T2 - 1.00796)
T3 = T7 / T3
IF(RN .LE. T3) GO TO 10
T4 = T7 / (3.63333 + T2 * (5.44444 + T2 - 4.66666))
T7 = 5 * T7
T2 = RN / T3
T3 = 2.1

T5 = 1.4
GO TO 30
T4 = T3 / (T4 + T5)
T7 = 5 * T3
T2 = RN
T5 = 1. - T5
T3 = 3. * T5
T5 = 2. * T5
T7 = 1. + T2 * (T2 + (2. * T7 + T4 - T3 + T2 * (T5 - T7 - T4) - T7))
RETURN

END
SUBROUTINE ISOS(U, V, W, KRN)  
C    SAMPLE A DIRECTION U, V, W ISOTROPICALLY.
C
10 T1=2. *RANF(KRN)-1.
20 T2=2. *RANF(KRN)-1.
30 RSQ=T1*T1+T2*T2
40 IF(RSQ.GT.1.0)GO TO 10
50 U=2. *RSQ-1.
60 T3=SQRT((1.-U**2)/RSQ)
70 V=T1*T3
80 W=T2*T3
90 RETURN
100 END

SUBROUTINE RDTAS(C, KRN, U, V, W, UOLD, VOLD, WOLD)  
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.
20 T2=2. *RANF(KRN)-1.
30 R=T1*T1+T2*T2
40 IF(R.GT.1.0)GO TO 10
50 R=SQRT((1.-C**2)/R)
60 T1=T1+R
70 T2=T2*R
80 IF(ABS(WOLD).GT.999999)GO TO 30
90 S=SQRT(1.-WOLD**2)
100 U=UOLD*C+(T1*UOLD*WOLD-T2*VOLD)/S
110 V=VOLD+C+(T1+VOLD+WOLD+T2*UOLD)/S
120 W=WOLD*C-T1*S
130 RETURN
140 30 U=T1
150 V=T2
160 W=WOLD*C
170 RETURN
180 END

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

END
program (p18 [p02])
begin
  clear (d14)
  NPC = NPC + 1
  IF (NPC GE NPPC) THEN
    set (d06)
    clear (d11)
  ENDIF
  suspend
end (p18)
APPENDIX E

Generated Parallel FORTRAN for HEP/UPX

(result of LGDF macro-expansion)
1 C--<<<<<<NON >>>>
2 C
3 PROGRAM GAMTEB
4 C
5 C------ Scalar Monte Carlo Transport Code
6 C
7 C----------------------- (HEPUNIX FORTRAN SYSTEM TABLES) -----------------------
8 COMMON /SYSTAB/ $LTR,$DW,$DR,LPR(40),LPS(40),LPX(40),LNL(40)
9 LOGICAL $LTR,$DW(15),$DR(15)
10 C
11 C----------------------- (FORTRAN TRACE TABLES) -----------------------
12 COMMON /TRCTAB/ LDM(15),LPM(40),PL,DL,PS,DS,DT,PT
13 CHARACTER*50 PL(0:18),DL(0:15)
14 CHARACTER*10 PS(0:18),DS(0:15)
15 CHARACTER*4 DT(15),PT(40)
16 C
17 LOGICAL GO
18 C
19 DT(1)= 'd01'
20 LDM(1)= 01
21 DT(2)= 'd09'
22 LDM(2)= 09
23 DT(3)= 'd15'
24 LDM(3)= 15
25 DT(4)= 'd07'
26 LDM(4)= 07
27 DT(5)= 'd03'
28 LDM(5)= 03
29 DT(6)= 'd04'
30 LDM(6)= 04
31 DT(7)= 'd05'
32 LDM(7)= 05
33 DT(8)= 'd06'
34 LDM(8)= 06
35 DT(9)= 'd02'
36 LDM(9)= 02
37 DT(10)= 'd11'
38 LDM(10)= 10
39 DT(11)= 'd11'
40 LDM(11)= 11
41 DT(12)= 'd12'
42 LDM(12)= 12
43 DT(13)= 'd06a'
44 LDM(13)= 06
45 DT(14)= 'd13'
46 LDM(14)= 13
47 DT(15)= 'd14'
48 LDM(15)= 14
49 C
50 PT(1)= 'top'
51 LPM(1)= -1
52 PT(2)= 'p00'
53 LPM(2)= 00
54 PT(3)= 'p10'
55 LPM(3)= 10
56 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)=p17t
LPM(30)=17
PT(31)=p17u
LPM(31)=17
PT(32)=p17v
169  PL(12)="init GAMTEB run parameters"
170  PS(13)="PCOUNT"
171  PL(13)="setup for particle counting"
172  PS(14)="REPORT"
173  PL(14)="format GAMTEB report"
174  PS(16)="RGENRAN"
175  PL(16)="generate random seed for next particle"
176  PS(17)="RNHIST"
177  PL(17)="run history for one particle and offspring"
178  PS(18)="CKCOMP"
179  PL(18)="check for run completion"
180  C
181  C  OPEN(4,FILE='tt')
182  C
183  I=1
184  7720 LPR(I)=0
185  LPX(I)=0
186  LPS(I)=0
187  LNL(I)=0
188  I=I+1
189  IF(I.LE.40) GO TO 7720
190  C
191  I=1
192  7730 CALL SETE($DW(I))
193  CALL SETE($DR(I))
194  CALL AWRITE($DW(I), .FALSE.)
195  CALL AWRITE($DR(I), .FALSE.)
196  CALL SETE($DW(I))
197  CALL SETE($DR(I))
198  I=I+1
199  IF(I.LE.15) GO TO 7730
200  C
201  GD=LREAD($LDR)
202  CALL CLOCK(ISTART)
203  CALL PTRACE(2.42, ISTART, 0.0)
204  CALL AWRITE($LDR, GD)
205  C
206  C---- START LGDF EXECUTION ----
207  C
208  CALL POO(2.1)
209  C
210  C---- AWAIT RESULTS ----
211  C
212  CALL AWRITE($DR(3), LREAD($DR(3)))
213  GD=LREAD($LDR)
214  CALL PTRACE(2.40, 0.0, 0.0)
215  C
216  CALL CLOCK(IEND)
217  CALL PTRACE(2.43, IEND-ISTART, 0.0)
218  CALL PTRACE(2.50, 0.0, 0.0)
219  C
220  I=2
221  7797 IF(LPX(I).GT.0) CALL PTRACE(1.51, LX(1), 0.0)
222  IF(LPS(I).GT.0) CALL PTRACE(1.52, LPS(I), 0.0)
223  IF(LNL(I).GT.0) CALL PTRACE(1.53, LNL(I), 0.0)
224  IF(LPR(I).EQ.-1) CALL PTRACE(1.54, 0.0, 0.0)
```
I=I+1
IF(I.LE.40) GO TO 7797
CALL PTRACE(2.55,0.0,0.0,0)
I=1
IF(FULL($DR(I))) THEN
   IF(VALUE($DR(I))) CALL PTRACE(2.56,1.0,0.0)
   IF (.NOT. VALUE($DR(I))) CALL PTRACE(2.57,1.0,0.0)
ENDIF
IF(I.LE.15) GO TO 7798
C     STOP
C     END
C     SUBROUTINE PTRACE(IPN, IFC, IDN, IVAL, RVAL)
C     COMMON /SYSTAB/ $LTR, $DR, LPR(40), LPM(40), LPX(40), LNL(40)
C     COMMON /TRCTAB/ LDM(15), LPM(40), PL, LS, DS, DT, PT
C     COMMON /TRCTAB/ LDM(15), LPM(40), PL, LS, DS, DT, PT
C     CHARACTER*50 PL(0:18), DL(0:15)
C     CHARACTER*10 PS(O:18), DS(0:15)
C     CHARACTER*4 DT(15), PT(40)
C     IF(IFC.EQ.40) WRITE(4,40) PT(IPN)
FORMAT(85,8F6.2) *** LODEF -- NORMAL TERMINATION ***
C     IF(IFC.EQ.41) WRITE(4,41) PT(IPN), DT(IDN), DS(LDM(IDN))
C     FORMAT(‘85,8F6.2,’ UNLATCHES ‘A4,’ ‘A10’
C     IF(IFC.EQ.42) WRITE(4,42) PT(IPN), IVAL
C     FORMAT(‘85,8F6.2,’ HEPCLOCK START TIME = ‘, I20,
C     + (100 NS CYCLES)’
C     IF(IFC.EQ.43) WRITE(4,43) PT(IPN), IVAL
C     FORMAT(‘85,8F6.2,’ HEPELAPSED TIME = ‘, I12,
C     + (100 NS CYCLES)’
C     IF(IFC.EQ.44) WRITE(4,44) PT(IPN), RVAL
C     FORMAT(‘85,8F6.2,’ VAX ELAPSED TIME = ‘, F6.2, ‘ SECS.’)
C     IF(IFC.EQ.45) WRITE(4,45) PT(IPN), RVAL
C     FORMAT(‘85,8F6.2,’ USER_TIME = ‘, F6.2, ‘ SECS.’
C     IF(IFC.EQ.46) WRITE(4,46) PT(IPN), RVAL
C     FORMAT(‘85,8F6.2,’ SYSTEM TIME = ‘, F6.2, ‘ SECS.’)
C     IF(IFC.EQ.47) WRITE(4,47) PT(IPN), PL(LPM(IPN)), PL(LPM(IPN)),
C     + PT(IPN), LPS(IPN), IVAL
C     IF(IFC.EQ.48) WRITE(4,48) PT(IPN), PS(LPM(IPN)), PT(IPN), IVAL
C     FORMAT(‘85,8F6.2,’ A4,‘, ‘A10, ‘ A50, ‘
C     + A4,’ ‘A10,’ ‘A50, ‘
C     IF(IFC.EQ.49) WRITE(4,49) PT(IPN)
C     FORMAT(‘85,8F6.2,’ ‘A4,’ ‘A10,’ ‘A50,’ ‘
C     + A4,’ ‘A50,’ ‘
C     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  52 IF(IFC.EQ.52) WRITE(4,52) PT(IPN),IVAL 
286  53 IF(IFC.EQ.53) WRITE(4,53) PT(IPN),IVAL 
287  54 FORMAT( '/', ' A4:', ', '+++ TERMINATED +++') 
288  55 IF(IFC.EQ.55) WRITE(4,55) PT(IPN) 
289  56 IF(IFC.EQ.56) WRITE(4,56) PT(IPN),DT(IDN),DS(LDM(IDN)) 
290  57 IF(IFC.EQ.57) WRITE(4,57) PT(IPN),DT(IDN),DS(LDM(IDN)) 
291  58 IF(IFC.EQ.58) WRITE(4,58) PT(IPN),IVAL 
292  59 FORMAT( '/', ' A4:', ', 'NO. LATCHES', ', I12) 
293  60 IF(IFC.EQ.59) WRITE(4,59) PT(IPN) 
294  61 IF(IFC.EQ.60) WRITE(4,60) PT(IPN),IVAL 
295  62 IF(IFC.EQ.61) WRITE(4,61) PT(IPN) 
296  63 IF(IFC.EQ.62) WRITE(4,62) PT(IPN),IVAL 
297  64 IF(IFC.EQ.63) WRITE(4,63) PT(IPN),IVAL 
298  65 IF(IFC.EQ.64) WRITE(4,64) PT(IPN),IVAL 
299  66 IF(IFC.EQ.65) WRITE(4,65) PT(IPN),IVAL 
300  67 IF(IFC.EQ.66) WRITE(4,66) PT(IPN),IVAL 
301  68 IF(IFC.EQ.67) WRITE(4,67) PT(IPN),IVAL 
302  69 IF(IFC.EQ.68) WRITE(4,68) PT(IPN),IVAL 
303  70 IF(IFC.EQ.69) WRITE(4,69) PT(IPN),IVAL 
304  71 IF(IFC.EQ.70) WRITE(4,70) PT(IPN),IVAL 
305  72 IF(IFC.EQ.71) WRITE(4,71) PT(IPN),IVAL 
306  73 IF(IFC.EQ.72) WRITE(4,72) PT(IPN),IVAL 
307  74 IF(IFC.EQ.73) WRITE(4,73) PT(IPN),IVAL 
308  75 IF(IFC.EQ.74) WRITE(4,74) PT(IPN),IVAL 
309  76 IF(IFC.EQ.75) WRITE(4,75) PT(IPN),IVAL 
310  77 IF(IFC.EQ.76) WRITE(4,76) PT(IPN),IVAL 
311  78 IF(IFC.EQ.77) WRITE(4,77) PT(IPN),IVAL 
312  79 IF(IFC.EQ.78) WRITE(4,78) PT(IPN),IVAL 
313  80 IF(IFC.EQ.79) WRITE(4,79) PT(IPN),IVAL 
314  81 IF(IFC.EQ.80) WRITE(4,80) PT(IPN),IVAL 
315  82 IF(IFC.EQ.81) WRITE(4,81) PT(IPN),IVAL 
316  83 IF(IFC.EQ.82) WRITE(4,82) PT(IPN),IVAL 
317  84 IF(IFC.EQ.83) WRITE(4,83) PT(IPN),IVAL 
318  85 IF(IFC.EQ.84) WRITE(4,84) PT(IPN),IVAL 
319  86 IF(IFC.EQ.85) WRITE(4,85) PT(IPN),IVAL 
320  87 IF(IFC.EQ.86) WRITE(4,86) PT(IPN),IVAL 
321  88 IF(IFC.EQ.87) WRITE(4,87) PT(IPN),IVAL 
322  89 IF(IFC.EQ.88) WRITE(4,88) PT(IPN),IVAL 
323  90 IF(IFC.EQ.89) WRITE(4,89) PT(IPN),IVAL 
324  91 IF(IFC.EQ.90) WRITE(4,90) PT(IPN),IVAL 
325  92 IF(IFC.EQ.91) WRITE(4,91) PT(IPN),IVAL 
326  93 IF(IFC.EQ.92) WRITE(4,92) PT(IPN),IVAL 
327  94 IF(IFC.EQ.93) WRITE(4,93) PT(IPN),IVAL 
328  95 IF(IFC.EQ.94) WRITE(4,94) PT(IPN),IVAL 
329  96 IF(IFC.EQ.95) WRITE(4,95) PT(IPN),IVAL 
330  97 IF(IFC.EQ.96) WRITE(4,96) PT(IPN),IVAL 
331  98 IF(IFC.EQ.97) WRITE(4,97) PT(IPN),IVAL 
332  99 IF(IFC.EQ.98) WRITE(4,98) PT(IPN),IVAL 
333 100 IF(IFC.EQ.99) WRITE(4,99) PT(IPN),IVAL 
334 101 IF(IFC.EQ.100) WRITE(4,100) PT(IPN),IVAL 
335 102 IF(IFC.EQ.101) WRITE(4,101) PT(IPN),IVAL 
336 103 IF(IFC.EQ.102) WRITE(4,102) PT(IPN),IVAL 

E-7
SUBROUTINE PO0(IPN, IPCXT)
            COMMON /SYSTAB/ $LTR, $DW, $DR, LPR(40), LPS(40),  LPX(40), LNL(40)
            LOGICAL $LTR, $DW(15), $DR(15)
            EXTERNAL P10
            LPX(IPN)=LPX(IPN)+1
            CALL CREATE(P10, 3, IPN)
            CALL PO1(4, IPN)
            RETURN
            END

SUBROUTINE PO1(IPN, IPCXT)
            COMMON /SYSTAB/ $LTR, $DW, $DR, LPR(40), LPS(40),  LPX(40), LNL(40)
            LOGICAL $LTR, $DW(15), $DR(15)
            EXTERNAL P12
            EXTERNAL P13
            EXTERNAL P14
            LPX(IPN)=LPX(IPN)+1
            CALL CREATE(P12, 5, IPN)
            CALL CREATE(P13, 6, IPN)
            CALL PO2(7, IPN)
            CALL CREATE(P14, 8, IPN)
            RETURN
            END

SUBROUTINE PO2(IPN, IPCXT)
            COMMON /SYSTAB/ $LTR, $DW, $DR, LPR(40), LPS(40),  LPX(40), LNL(40)
            LOGICAL $LTR, $DW(15), $DR(15)
            EXTERNAL P16
            EXTERNAL P17
            EXTERNAL P18
            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)
393 CALL CREATE(P17, 22, IPN)
394 CALL CREATE(P17, 23, IPN)
395 CALL CREATE(P17, 24, IPN)
396 CALL CREATE(P17, 25, IPN)
397 CALL CREATE(P17, 26, IPN)
398 CALL CREATE(P17, 27, IPN)
399 CALL CREATE(P17, 28, IPN)
400 CALL CREATE(P17, 29, IPN)
401 CALL CREATE(P17, 30, IPN)
402 CALL CREATE(P17, 31, IPN)
403 CALL CREATE(P17, 32, IPN)
404 CALL CREATE(P17, 33, IPN)
405 CALL CREATE(P17, 34, IPN)
406 CALL CREATE(P17, 35, IPN)
407 CALL CREATE(P17, 36, IPN)
408 CALL CREATE(P17, 37, IPN)
409 CALL CREATE(P17, 38, IPN)
410 CALL CREATE(P17, 39, IPN)
411 CALL CREATE(P18, 40, IPN)
412 RETURN
413 END
414 C==>END: p00.f
C--<<<<p10.f>>>  
C------ SETUP -- set up problem constants  
C     SUBROUTINE PIO(IPN, IPCTX)  
C     COMMON /SYSTAB/ SLTR, SDW, SDR, LPR(40), LPS(40), LPX(40), LNL(40)  
C     LOGICAL SLTR, SDW(15), SDR(15)  
C-> do3: constants -- problem constants  
C     COMMON /DO3/ E, RHO, CL, CL2, CRAD, CRAD2, WCP1, WCP2, EC, FIM  
C-> do4: cs tables -- cross section tables  
C     COMMON /DO4/ EL, XC, XPP, XPE  
C-> do5: src values -- source values  
C     COMMON /DO5/ GIERS, GIWT, GIU, GIV, GIW, GIX, GIY, GIZ, IGIA  
C     INTEGER IGIA  
C     LOGICAL GO, DFPROG  
IF(LPR(IPN).EQ.-1) RETURN  
7799 DFPROG=.FALSE.  
CALL AWRITE(SDW(5), GO)  
CALL AWRITE(SDW(6), LREAD(SDW(5)))  
CALL AWRITE(SDW(7), LREAD(SDW(6)))  
GO=LREAD(*DW(7))  
7798 LPX(IPN)=LPX(IPN)+1  
LPR(IPN)=1  
LPR(IPCTX)=LPR(IPCTX)+1  
CL=20.0  
CL2=CL+10.  
CRAD=1.0  
CRAD2=CRAD**2  
WCP1=.5  
WCP2=.25  
EC = .001  
FIM(1)=1.0  
FIM(2)=2.0  
CALL AWRITE(*DW(5), .FALSE.)  
DFPROG=.TRUE.  
CALL AWRITE(*DR(5), .FALSE.)  
DFPROG=.TRUE.  
CONVERT CROSS-SECTION UNITS TO BE PER CM.  
DO 1 I=1,35  
X(I)=ALOG(X(I)*RHO)  
1 IF(XPP(I).EQ.0.) XPP(I)=1.0E-37  
IF(XPE(I).EQ.0.) XPE(I)=1.0E-37  
XPP(I)=ALOG(XPP(I)*RHO)  
XPE(I)=ALOG(XPE(I)*RHO)  

E-10
EL(1)=ALOG(E(1))

1 CONTINUE

CALL AWRITE(*DW(6),.FALSE.)

DFPROG= TRUE.

CALL AWRITE(*DR(6),.FALSE.)

DFPROG= TRUE.

C

SET SOURCE VALUES

C

GIEQ = 6.0

GIWT = 1.0

GIU = 0.0

GIV = 1.0

GIW = 0.0

GIX = 0.0

GIY = 0.000001

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=LREAD(*LTR)

CALL PTRACE(IPN,48,0,LPX(IPN),0.0)

CALL AWRITE(*LTR,VALUE(*LTR))

LPR(IPN) = -1

RETURN

ENDIF

GO TO 7797

C=>END pio f

BLOC DATA

D03: constants - problem constants

COMMON /D03/ E,RHD,CL,CL2,CRAD,CRAD2,WCP1,WCP2,EC,FIM

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

D04: cs tables - cross section tables

COMMON /D04/ EL,XC,XPP,XPE

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

D02/RH0/2,22/

DATA (E(I),I=1,35)/ 0.001, 0.0015, 0.002, 0.003, 0.004, 0.005, 0.006, 0.008, 0.01, 0.015, 0.02, 0.03, 0.04, 0.05, 0.06, 0.08, 0.1, 0.15, 0.2, 0.3, 0.4, 0.5, 0.6, 0.8, 1.1, 1.5, 2.0, 3.0, 4.0, 5.0, 6.0, 8.0, 10.0, 15.0, 20.0/

DATA (XC(I),I=1,35)/ 0.0150, 0.0296, 0.0451, 0.0717, 0.0913

DATA (XPP(I),I=1,35)/ 0.00316, 0.00923, 0.0152, 0.0208, 0.0256, 0.0343.

DATA (XPE(I),I=1,35)/ 0.00414, 0.00547, 0.00652/
<table>
<thead>
<tr>
<th>DATA (XPE(I),I=1,35)</th>
<th>2010</th>
<th>632</th>
<th>280</th>
<th>87.7</th>
<th>37.3</th>
<th>18.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10.4</td>
<td>4.01</td>
<td>1.91</td>
<td>0.489</td>
<td>0.192</td>
<td>0.0491</td>
</tr>
<tr>
<td>2</td>
<td>0.00481</td>
<td>0.00179</td>
<td>0.000862</td>
<td>0.000234</td>
<td>0.0000918</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td></td>
</tr>
<tr>
<td>END</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
PARAM -- init GAMTEN run parameters

SUBROUTINE P12(IPN, IPCTXT)

--- (HEPUNIX FORTRAN SYSTEM TABLES) ---

COMMON /SYSTAB/ *LTR, *DW, *DR, LPR(40), LPS(40), LPX(40), LNL(40)
LOGICAL *LTR, *DW(15), *DR(15)

COMMON /D01/ ID01
INTEGER ID01

COMMON /D09/ CL1, CL2, CRAD, CRAD2, WCP1, WCP2, EC
REAL CL1, CL2, CRAD, CRAD2, WCP1, WCP2, EC

COMMON /D15/ ID015
INTEGER ID015

COMMON /D06/ IGCOL, IGCO, GWCO, GWRL, IGNR, GWRG, IGCUTF,
+ GABSOR, IGNS, TRANS(35), ESCAPE(35), BSCAT(35),
+ BTOT2, TTDT2, ETOT2, BTOT, TTOT
REAL GWCO, GWRL, GWRG, GABSOR

COMMON /D02/ NPP
INTEGER NPP
LOGICAL GO, DFPROG

IF(LPR(IPN), EQ, -1) RETURN

CALL AWRITE(*DW(3), GO)
CALL AWRITE(*DW(8), LREAD(*DW(3)))
CALL AWRITE(*DW(9), LREAD(*DW(8)))
GO=LREAD(*DW(9))

LPR(IPN)=1
LPR(IPCTXT)=LPR(IPCTXT)+1

GO=LREAD(*LTR)
WRITE(6, *) 'P12: ENTER NO. OF PARTICLES TO RUN.'
NPP=500000
READ(5, *) NPP
CALL AWRITE(*LTR, GO)
IF (NPP, EQ, 0) THEN
CALL AWRITE(*DW(3), FALSE.)

E-13
57         DFPROG= .TRUE.
58         CALL AWRITE(*DR(3), FALSE.)
59         DFPROG= .TRUE.
60         GO TO 7797
61      ELSE
62         CALL AWRITE(*DW(9), FALSE.)
63         DFPROG= .TRUE.
64         CALL AWRITE(*DR(9), FALSE.)
65         DFPROG= .TRUE.
66      ENDIF
67      C
68      IGNCOL=0
69      IGNCO=0
70      QWCO=0
71      QCWP=0
72      QWRL=0.
73      QWRG=0.
74      QABSR=0.
75      IGN=0
76      IGCUTF=0
77      IGNS=0
78      DO 5 I = 1, 35
79          TRANS2(I)= 0.0
80          BSCAT2(I)= 0.0
81          ESCAPE2(I)= 0.0
82          TRANS2(I)= 0.0
83          BSCAT2(I)= 0.0
84          ESCAPE2(I)= 0.0
85      5 CONTINUE
86      BTOT=0.0
87      BTOT2=0.0
88      TTOT=0.0
89      TTOT2=0.0
90      ETOT=0.0
91      ETOT2=0.0
92      CALL AWRITE(*DW(B), FALSE.).
93      DFPROG= .TRUE.
94      CALL AWRITE(*DR(B), FALSE.).
95      DFPROG= .TRUE.
96      GO TO 7797
97      7797 IF (.NOT. DFPROG) THEN
98          GO=LAREAD(SLTR)
99      I;ALCPTRACE(LPNI;#01 LPXI;PN)D)
100      CALL AWRITE(SLTR, VALUE(*LTR))
101      LPR(LPNI)= -1
102      RETURN
103      ENDIF
104      GO TO 7799
105      END
106      C==END.pl2.f
SUBROUTINE P13(IPN, IPCTXT)

COMMON /SYSTAB/ $LTR, $DR, LPR(40), LPS(40), LPX(40), LNL(40)
LOGICAL $LTR, $DR(15)

INTEGER NPP
COMMON /D02L/ NPP

INTEGER NPPR
COMMON /D10/ NPPR

INTEGER NPC, NPPC
COMMON /D11/ NPC, NPPC

INTEGER NPPS, NPS, KRN2
COMMON /D12/ NPPS, NPS, KRN2

LOGICAL GO, DFPROG

IF(LPR(IPN).EQ. -1) RETURN

DFPROG= .FALSE.
CALL AWRITE($DR(9), LAREAD($DR(9)))
CALL AWRITE($SW(10), GO)
CALL AWRITE($SW(11), LAREAD($SW(10)))
CALL AWRITE($SW(12), LAREAD($SW(11)))
GO=LAREAD($SW(12))
LPX(IPN)=LPX(IPN)+1
LPR(IPN)=1
LPR(IPCTXT)=LPR(IPCTXT)+1

KRN = 123454321;

CALL AWRITE($SW(12), .FALSE.)
DFPROG= .TRUE.
CALL AWRITE($SW(12), .FALSE.)
DFPROG= .TRUE.

NPC=0
NPPC=NPP

CALL AWRITE($SW(11), .FALSE.)
DFPROG= .TRUE.
CALL AWRITE($SW(11), .FALSE.)
DFPROG= .TRUE.

C -- MAKE COPY OF NPP FOR REPORT PROCESS
NPRR-NPP

CALL AWRITE(*D(10), .FALSE.)
DFPROG= .TRUE.
CALL AWRITE(*D(10), .FALSE.)
DFPROG= .TRUE.
GO=LREAD(*D(9))
GO=LREAD(*D(9))
DFPROG= .TRUE.
GO TO 7797
7797 IF(.NOT. DFPROG) THEN
GO=LREAD(*LTR)
CALL PTRACE(IPN, 4B, 0, LPX(IPN), 0, 0)
CALL AWRITE(*LTR, VALUE(*LTR))
LPR(IPN)= -1
RETURN
END IF
GO TO 7799
END
C==>END:p13.f
E-16
C--- <<<p14.f>>> 

C REPORT -- format GAMTEB report

SUBROUTINE P14(INP, IPTXT)

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--> d06: run stats -- run statistics
COMMON /D06/ IGNCD, IGNCD, GWCD, GWCP, GWRL, IGNR, GWRG, IGCUF,
+ GABSOR, IGNS,
+ BTOT2, TTOT2, ETOT2, BTOT, TTOT

C--> d10: NPPR -- particle count for report
COMMON /D10/ NPPR
INTEGER NPPR

C--> d03: constants -- problem constants
COMMON /D03/ E, RH0, CL, CL2, CRAD, CRAD2, WCP1, WCP2, EC, FIM
REAL E(35), RH0, CL, CL2, CRAD, CRAD2, WCP1, WCP2, EC, FIM(2)

C--> d07: report -- GAMTEB report file
COMMON /D07/ ID07
INTEGER ID07
LOGICAL GO, DFPRO

REAL RTRANS(35), RSCAT(35), RESCAP(35)
IF(LPR(INP), EQ. -1) RETURN
7799 DFPRO= FALSE
CALL AWRITE($DR(13), LAREAD($DR(13)))
CALL AWRITE($DR(10), LAREAD($DR(10)))
CALL AWRITE($DR(5), LAREAD($DR(5)))
7798 LPX(INP)=LPX(INP)+1
LPR(INP)=1
LPR(IPTXT)=LPR(IPTXT)+1

C PRINT OUTPUT

C NPP AND NPS ARE USED INTERCHANGEABLY IN THE ORIGINAL CODE
C 140 NPS = NPS - 1
C 140 NPS=NPPR

C FIX UP NAME ALIAS FOR NPP
GO=LAREAD($LTR)
WRITE(4,7634) IGNCD
7634 FORMAT(5H NCOL,110)
WRITE(4,1401)

E-17
E-18
Apr 30 14:10 1985 p14.f Page 3

113   DO 225 I=1,35
114      WRITE(4,210) E(I),ESCAPE(I),RESCAP(I)          RENAME
115   225 CONTINUE
116      WRITE(4,2020) ETDT, RETDT
117      WRITE(4,202)
118      202 FORMAT(/,1X,1HE,13X,9HTRANS,9X,9HREL ERROR)
119   DO 230 I=1,35
120      WRITE(4,210) E(I),TRANS(I),RTRANS(I)
121   210 FORMAT(5X,1PE10.3,5X,1PE10.3,5X,OPF6.3)
122     230 CONTINUE
123      WRITE(4,2020) TTDT, RTDT
124     QABSOR = GABSOR/NPS
125    GWRG=GWRG/NPP
126    GWRL=GWRL/NPP
127    GWCP=GWCP/NPP
128    GWCO=GWCO/NPP
129    WRITE(4,221) GABSOR, IGCTF
130   221 FORMAT(/,1X,9HABSOR= ,1PE10.3,9HCUTOFF = ,15)
131     WRITE(4,3728) IGN5, IGNR
132    3728 FORMAT(2BH TRACKS CREATED BY SPLITTING,1B,
133           1 24H TRACKS LOST TO ROULETTE,1B)
134     WRITE(4,3729) GWRG, GWRL
135    3729 FORMAT(27H WEIGHT CREATED BY ROULETTE,1PE11.4,
136           1 24H WEIGHT LOST TO ROULETTE,1PE11.4)
137     WRITE(4,3730) IGNCO
138    3730 FORMAT(29H TRACKS LOST TO WEIGHT CUTOFF,1B)
139     WRITE(4,3731) GWCP, GWCO
140    3731 FORMAT(32H WEIGHT CREATED BY WEIGHT CUTOFF,1PE11.4,
141           1 29H WEIGHT LOST TO WEIGHT CUTOFF,1PE11.4)
142     WRITE(4,2021) TEND
143   2021 FORMAT(/,13HTOTAL_TIME = ,1PE10.3, 8H SECONDS)
144   C -- RETURN ACCESS TO I/O
145     CALL AWRITE($LTR.GO)
146      GO=LAREAD($DR(13))
147      GO=LAREAD($DW(13))
148      DFPROC= TRUE.
149      GO=LAREAD($DR(10))
150      GO=LAREAD($DW(10))
151      DFPROC= TRUE.
152      GO TO 7797
153    7797 IF (.NOT. DFPROC) THEN
154      GO=LAREAD($LTR)
155     CALL PTRACE(IPN,48.0,LPX(IPN),0.0)
156     CALL AWRITE($LTR.VALUE($LTR))
157      LPR(IPN)=-1
158     RETURN
159   ENDIF
160      GO TO 7799
161   END
162 C=>END: p14.f

E-19
C---GENRAN -- generate random seed for next particle

SUBROUTINE P16(IPN, IPCTXT)

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-> 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 DFPROG

IF(LPR(IPN).EQ.-1) RETURN
CALL AWRITE(DR(12), LAREAD(DR(12)))
GO=LAREAD(DW(14))
LPX(IPN)=LPX(IPN)+1
LPR(IPN)=1
LPR(IPCTXT)=LPR(IPCTXT)+1

C START A HISTORY
10 NPS = NPS + 1
IF(NPS.GT.NPPS) THEN
GO TO 140
Go=LAREAD(DW(12))
DFPROG= TRUE.
ELSE
C GENERATE NEW RANDOM SEED
XJUNK = RANDO(KRN2)
KERN = KRN2
CALL AWRITE(DW(14), FALSE.)
DFPROG= TRUE.
ENDIF
GO TO 7797
7797 IF(.NOT.DFPROG) THEN
GO=LAREAD(LTR)
CALL PTRACE(IPN, 48, O, LPX(IPN), O, O)
CALL AWRITE(LTR, VALUE(LTR))
LPR(IPN) = -1
RETURN
ENDIF
GO TO 7799
END
C---END: p16.f
### REAL FUNCTION RAND0(KERN)

58  REAL FUNCTION RAND0(KERN)  
59  KERN = MOD(1+7421*KERN,131072)  
60  4 FORMAT(1X,F12.8)  
61  RAND0 = FLOAT(KERN)/131072.  
62  RETURN  
63  END
C-<<<<p17.f>>>>

C---- RNIST -- run history for one particle and offspring
C
SUBROUTINE P17(IPN, IPCTXT)

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
COMM /D06/ IGNCOL, IGNCO, GWCD, GWCP, GWRL, IGNR, GWRG, IGCUF,
+ GABSOR, IGNS,
+ TRANS, ESCAPE, BSCAT, TRAN2, ESCAP2, BSCAT2,
+ BTOT2, TTOT2, EOTOT, BROT, TTOT
INTEGER IGNCOL, IGNCO, GWCD, GWCP, GWRL, IGNR, GWRG, IGCUF, IGNS
C
C--- d03: constants -- problem constants
COMM /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
C--- d04: cs tables -- cross section tables
COMM /D04/ EL, XC, XPP, XPE
REAL EL(35), XC(35), XPP(35), XPE(35)
C
C--- d05: src values -- source values
COMM /D05/ GIERG, GIWT, GIU, GIW, GIX, GIY, GIZ, IGIA
REAL GIERG, GIWT, GIU, GIW, GIX, GIY, GIZ
C
C--- d14: signal -- particle history completion signal
COMM /D14/ ID14
INTEGER ID14
C
C-- SCALAR MONTE CARLO CODE TO TRANSPORT 0.01 TO 20.0 MEV
C
C GAMMA RAYS IN A CARBON CYLINDER OF LENGTH CL, RADIUS CRAD

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, EOTOT
INTEGER INBNK, NBANK
INTEGER KRN
INTEGER CUTOFF
IF (LPR(IPN).EQ. -1) RETURN

7799 DFPROG=.FALSE.

E-22
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($DR(15),GO)
 GO=LAREAD($DR(15))
 7798 LPX(IPN)=LPX(IPN)+1
  LPR(IPCTXT)=LPR(IPCTXT)+1
 CALL AWRITE($DR(14),GO)
 GO=LAREAD($DR(14))
 KRN=KERN
 GO=LAREAD($DR(14))
 DFPROG= TRUE.
 C -- MAKE LOCAL COPIES OF INITIAL SOURCE VALUES
 ERG=GIERG
 WT=GINT
 U=GIV
 V=GIW
 X=GIX
 Y=GIY
 Z=GIZ
 IA=IGIA
 NCOL=0
 NCO=0
 WC0=0
 WRL=0.
 WRG=0.
 NR=0
 CUTOFF=0
 ABSORB=0
 DD 5 I = 1.35
 BSCAT(I)=0.
 TRANSI(I)=0.
 ESCAPI(I)=0.
 5 CONTINUE
 BTOT=0.
 ITOT=0.
 ETOT=0.
 INBNK=0
 C -- INITIAllZE LOCAL PARTICLE BANK INDEXES
 NBANK=0
 INBNK=0
 C
 CALCULATE DISTANCE DLS TO NEXT SURFACE INTERSECTION
 FOR ALL THREE SURFACES AND ALSO THE NUMBER JA OF THE
 NEXT SURFACE INTERSECTED
CALL TRACK(IA, JA, X, Y, Z, U, V, W, CL, CRAD2, DLS)

DO 30 IE = 1, 35
   IF(ERG.GT.E(I)) GO TO 30
   I = IE
   GO TO 31
30 CONTINUE

INTERPOLATION TO GET CROSS SECTIONS AS F(ERG)

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( XP(I-1)+F*(XP(I)-XP(I-1)) )
   XSPE = EXP( XPE(I-1)+F*(XPE(I)-XPE(I-1)) )
   XST = XSC + XSPP + XSPE

CALCULATE DISTANCE TO NEXT COLLISION

S = -(ALOG(RANF(KRN))/XST

SEE IF COLLISION IS STILL INSIDE CYLINDER

IF S.LT. DLS GO TO 60

TRANSI(1) = TRANSI(1) + WT

TTOTI = TTOTI + WT

ETOTI = ETOTI + WI

GO TO 11

GO TO(42, 50, 53, 52) JA

BSCATI(1) = BSCATI(1) + WT

BTOTI = BTOTI + WT

GO TO 11

52 TRANSI(1) = TRANSI(1) + WT

TTOTI = TTOTI + WT

GO TO 11

50 ESCAPI(1) = ESCAPI(1) + WT

ETDII = ETDII + WI

GO TO 11

CROSS INTERNAL SURFACE SPLIT OR ROULETTE

IAP = IA

IA = 2 - IA/2

T = FIM(IA)/FIM(IAP)

IF(T.LT.RANF(KRN)) GO TO 56

WTSAV = WT

WRG = WRG + (WT-WTSAV)

GO TO 20

KILLED IN RUSSIAN ROULETTE

WRL = WRL + WT

NR = NR + 1

GO TO 11

SPLITTING

NP = TI - 1

WT = WT/TI

E-24
DO 59 IX=1,8
59 BANK(INBNK, IX)=PBL(IX)  BUGFIX
DO 61 IX=1,2
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=1,2
522 IPBL(IX)=IBANK(INBNK, IX)
NBANK=NBANK+1
INBNK=INBNK+1
GO TO 20

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
832 CONTINUE
206 IF(RANF(KRN) GE XSC/XSTSB) GO TO 100
207 T1 = 1.956917*ERG
208 C GET NEW ENERGY TSC AND COMPTON SCATTERING ANGLE
209 CALL KLEIN(T1, T4, KRN)  ARGUMENTS
210 CSA = .511008*T4
212 IF(ABS(CSA) GT 1.) CSA=SIGN(1.,CSA)
213 ERG = T5

C SEE IF NEW ENERGY IS LESS THAN CUTOFF
216 IF(ERG GT EC) GO TO 70
217 CUTOFF = CUTOFF + 1
GO TO 11

C MAKE COMPTON ANGLE RELATIVE TO PROBLEM COORDINATE SYSTEM
70 UOLD = U
VOLD = V
WOLD = W
223 CALL ROTAS(CSA, KRN, U, V, W, UOLD, VOLD, WOLD)  ARGUMENTS
GO TO 20

E-25
PAIR PRODUCTION

100 ERG = 0.511008
WT = 2.*WT

CHECK ENERGY CUTOFF
IF(ERG.GT.EC) GO TO 110
CUTOFF = CUTOFF + 1
GO TO 11

ISOTROPIC EMISSION IN LAB SYSTEM
110 CALL ISOS(U,V,W,WRN) ARGs
GO TO 20

PHOTOELECTRIC ABSORPTION
NOW HANDLED BY SURVIVAL BIASING
C 130 ABSORB = ABSORB + WT
GO TO 11

TERMINATE PARTICLE TO WEIGHT.CUTOFF
642 WC0=WC0+WT
NCO=NCO+1
GO TO 11

GET EXCLUSIVE UPDATE ACCESS TO RUN STATISTICS
234 GO=LREAD(*DR(B))
IGNCOL=IGNCOL+NCOL
IGNCO=IGNCO+NCO
GWCO=GWCO+WC0
GWC0=GWC0+WCP
GWR0=GWR0+WRL
GWR=GWCR+WRC
IGNR=IGNR+NR
ICUTF=ICUTF+CUTOFF
GABSOR=GABSOR+ABSOR
IGNS=IGNS+NS
DO 829 I=1,35
BSCAT(I)=BSCAT(I)+BSCAT(I)
BSCAT2(I)=BSCAT2(I)+BSCAT2(I)*2
TRANS(I)=TRANS(I)+TRANS(I)
TRANS2(I)=TRANS2(I)+TRANS2(I)*2
ESCAPE(I)=ESCAPE(I)+ESCAPI(I)
ESCAP2(I)=ESCAP2(I)+ESCAP2(I)*2
829 CONTINUE

B0T=BIT+B0T0
T0T=T0T+T0T0
ET0T=ET0T+ET0T0
BT0T2=BT0T2+BT0T2
TT0T2=TT0T2+TT0T2
ET0T2=ET0T2+ET0T2

GIVE BACK UPDATE ACCESS TO RUN STATISTICS
271 CALL AWRITE(*DR(9),.FALSE.)
272 DFP0G=.TRUE.
273 C -- SIGNAL COMPLETION OF THIS PARTICLE HISTORY
277 CALL AWRITE(*DW(15),.FALSE.)
278 DFP0G=.TRUE.
279 CALL AWRITE(*DR(15),.FALSE.)
280 DFP0G=.TRUE.
C GO TO 10
GO TO 7797
7797 IF (.NOT. DFPROG) THEN
GO=LAREAD(*LTR)
CALL PTRACE(IPN, 48, O, LPX(IPN), O, O)
CALL AWRITE(*LTR, VALUE(*LTR))
LPR(IPN)=-1
RETURN
END IF
GO TO 7799
END
C===>END
SUBROUTINE TRACK(IA, JA, X, Y, Z, U, V, W, CL, CL2, CRAD2, DLS)
ARGS
C CALCULATE ALL INTERSECTIONS WITH ALL THREE SURFACES
DLS = 1.0E10
IF (IA.EQ.2) GO TO 19
D0 300 J=1,3
D1 = -1.0
GO TO 280
55 IF (V.EQ.0.) GO TO 300
D1 = (-Y/V)
GO TO 280
D2 = -2.*A1
IF (J.EQ.JA) THEN NEW
D1=-2.*A1
NEW
GO TO 290
END IF
D0 280
D2 = -D1
IF (D2.GT.0.) D1=D2
D1 = (CL-Y)/V
GO TO 281
END
C SAMPLE FROM KLEIN-NISHINA USING INVERSE FIT

RN=RNF(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
T7=1.65899+T2*(1.62937*T2-1.00796)
T3=T7/T3
IF(RN.LE.T3)GO TO 10
T4=(T6-1.20397)/T3
T7=3*EXP(T4*(T3-RN))
GO TO 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
SUBROUTINE ISOS(U, V, W, KRN)

SAMPLE A DIRECTION U, V, W ISOTROPICALLY.

C T5 = 1.0 - T5
C T3 = 2.0 * T3
C T7 = 1.0 + 2.0 * (T2 * (2.0 * T4 + T3 * (T5 - T4)) - T7)
C T4 = T7 * T1

RETURN

END

SUBROUTINE ROTAS(C, KRN, U, V, W, UOLD, VOLD, WOLD)

ROTATE UOLD, VOLD, WOLD TO U, V, W THROUGH A POLAR ANGLE WHOSE COSINE IS C, AND THROUGH AN AZIMUTHAL ANGLE SAMPLED UNIFORMLY.

C T1 = 2.0 * RANF(KRN) - 1.
C T2 = 2.0 * RANF(KRN) - 1.
C R = T1 ** 2 + T2 ** 2
C IF(R.GT.1.0) GO TO 10
C U = 2.0 * RANF(KRN) - 1.
C T3 = SQRT((1.0 - T1 ** 2) / R)
C V = T1 * T3
C W = T2 * T3
C RETURN

END

REAL FUNCTION RANF(KRN)

KERN = MOD(1+9621*KERN,131072)
RANF = FLOAT(KERN)/131072.
RETURN

END
C---- CKCOMP -- check for run completion
C
SUBROUTINE P1B(IPN, IPCTXT)

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, GWRG, IGNR, IGNS, IGCUF,
+ GABSOR, IGN.
+ TRANS, ESCAPE, BSCAT, TRANS2, ESCAP2, BSCAT2,
+ BTOT2, ETOT2, ETOT, BTOT, TTOT
INTEGER IGNCOL, IGNCO, IGNR, GABSOR, IGN,
+ TRANS(35), ESCAPE(35), BSCAT(35),
+ TRANS2(35), ESCAP2(35), BSCAT2(35),
+ BTOT2, ETOT2, ETOT, BTOT, TTOT

C-> d11: NPC:NPCC -- particle completion control
COMMON /D11/ NPC, NPCC
INTEGER NPC, NPCC

C-> d14: signal -- particle history completion signal
COMMON /D14/ NPCC
INTEGER NPCC

LOGICAL GO, DFPROG

IF(LPR(IPN).EQ.-1) RETURN
DFPROG=.FALSE.
IF(LNL(IPN).GT.0) THEN
IF(FULL(*DR(B))) THEN
CALL AWRITE(*DW(13), GO)
GO=LAREAD(*DW(13))
GO=LAREAD(*DR(B))
DFPROG=.TRUE.
LNL(IPN)=LNL(IPN)-1
ENDIF
ENDIF
GO TO 7797
ENDIF

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(B), LAREAD(*DR(B)))
CALL AWRITE(*DR(11), LAREAD(*DR(11)))
CALL AWRITE(*DR(15), LAREAD(*DR(15)))
CALL AWRITE(*DR(B), LAREAD(*DR(B)))
CALL AWRITE(*DW(13), GO)
GO=LAREAD(*DW(13))
LPX(IPN)=LPX(IPN)+1
LPR(IPN)=1
57 LPR(IPCTX)+1
58
59 C
60 GO=LAREAD(*DR(15))
61 GO=LAREAD(*DW(15))
62 DFPROG=. TRUE.
63 NPC=NPC+1
64 IF (NPC.GE.NPPC) THEN
65 CALL AWRITE(*DW(13),.FALSE.)
66 DFPROG=. TRUE.
67 CALL AWRITE(*DR(13),.FALSE.)
68 DFPROG=. TRUE.
69 LNL(IPN)=LNL(IPN)+1
70 GO=LAREAD(*DR(11))
71 GO=LAREAD(*DW(11))
72 DFPROG=. TRUE.
73 ENDIF
74 GO TO 7797
75 7797 IF (.NOT. DFPROG) THEN
76 GO=LAREAD(*LTR)
77 CALL PTTRACE(IPN,48,0,LPX(IPN),0,0)
78 CALL AWRITE(*LTR,VALUE(*LTR))
79 LPR(IPN)= -1
80 RETURN
81 ENDIF
82 GO TO 7799
83 END
84 C==>END: p18.f
APPENDIX F

Reference Output for 100 Particles
<table>
<thead>
<tr>
<th>E</th>
<th>BSCAT</th>
<th>REL ERROR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.000e-03</td>
<td>0</td>
<td>e+00</td>
</tr>
<tr>
<td>1.500e-03</td>
<td>0</td>
<td>e+00</td>
</tr>
<tr>
<td>2.000e-03</td>
<td>0</td>
<td>e+00</td>
</tr>
<tr>
<td>3.000e-03</td>
<td>0</td>
<td>e+00</td>
</tr>
<tr>
<td>4.000e-03</td>
<td>0</td>
<td>e+00</td>
</tr>
<tr>
<td>5.000e-03</td>
<td>0</td>
<td>e+00</td>
</tr>
<tr>
<td>6.000e-03</td>
<td>0</td>
<td>e+00</td>
</tr>
<tr>
<td>8.000e-03</td>
<td>0</td>
<td>e+00</td>
</tr>
<tr>
<td>1.000e-02</td>
<td>0</td>
<td>e+00</td>
</tr>
<tr>
<td>1.500e-02</td>
<td>0</td>
<td>e+00</td>
</tr>
<tr>
<td>2.000e-02</td>
<td>0</td>
<td>e+00</td>
</tr>
<tr>
<td>3.000e-02</td>
<td>0</td>
<td>e+00</td>
</tr>
<tr>
<td>4.000e-02</td>
<td>0</td>
<td>e+00</td>
</tr>
<tr>
<td>5.000e-02</td>
<td>0</td>
<td>e+00</td>
</tr>
<tr>
<td>6.000e-02</td>
<td>0</td>
<td>e+00</td>
</tr>
<tr>
<td>8.000e-02</td>
<td>0</td>
<td>e+00</td>
</tr>
<tr>
<td>1.000e-01</td>
<td>0</td>
<td>e+00</td>
</tr>
<tr>
<td>1.500e-01</td>
<td>0</td>
<td>e+00</td>
</tr>
<tr>
<td>2.000e-01</td>
<td>0</td>
<td>e+00</td>
</tr>
<tr>
<td>3.000e-01</td>
<td>0</td>
<td>e+00</td>
</tr>
<tr>
<td>4.000e-01</td>
<td>0</td>
<td>e+00</td>
</tr>
<tr>
<td>5.000e-01</td>
<td>0</td>
<td>e+00</td>
</tr>
<tr>
<td>6.000e-01</td>
<td>0</td>
<td>e+00</td>
</tr>
<tr>
<td>8.000e-01</td>
<td>0</td>
<td>e+00</td>
</tr>
<tr>
<td>1.000e+00</td>
<td>0</td>
<td>e+00</td>
</tr>
<tr>
<td>1.500e+00</td>
<td>0</td>
<td>e+00</td>
</tr>
<tr>
<td>2.000e+00</td>
<td>0</td>
<td>e+00</td>
</tr>
<tr>
<td>3.000e+00</td>
<td>0</td>
<td>e+00</td>
</tr>
<tr>
<td>4.000e+00</td>
<td>0</td>
<td>e+00</td>
</tr>
<tr>
<td>5.000e+00</td>
<td>0</td>
<td>e+00</td>
</tr>
<tr>
<td>6.000e+00</td>
<td>0</td>
<td>e+00</td>
</tr>
<tr>
<td>8.000e+00</td>
<td>0</td>
<td>e+00</td>
</tr>
<tr>
<td>1.000e+01</td>
<td>0</td>
<td>e+00</td>
</tr>
<tr>
<td>1.500e+01</td>
<td>0</td>
<td>e+00</td>
</tr>
<tr>
<td>2.000e+01</td>
<td>0</td>
<td>e+00</td>
</tr>
</tbody>
</table>

TOTAL: 0 e+00 0
<table>
<thead>
<tr>
<th></th>
<th>TRANS</th>
<th>REL ERROR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000e-03</td>
<td>0.000e+00</td>
<td>0.000e+00</td>
</tr>
<tr>
<td>1.000e-02</td>
<td>0.000e+00</td>
<td>0.000e+00</td>
</tr>
<tr>
<td>1.500e-02</td>
<td>0.000e+00</td>
<td>0.000e+00</td>
</tr>
<tr>
<td>2.000e-02</td>
<td>0.000e+00</td>
<td>0.000e+00</td>
</tr>
<tr>
<td>3.000e-02</td>
<td>0.000e+00</td>
<td>0.000e+00</td>
</tr>
<tr>
<td>4.000e-02</td>
<td>0.000e+00</td>
<td>0.000e+00</td>
</tr>
<tr>
<td>5.000e-02</td>
<td>0.000e+00</td>
<td>0.000e+00</td>
</tr>
<tr>
<td>6.000e-02</td>
<td>0.000e+00</td>
<td>0.000e+00</td>
</tr>
<tr>
<td>8.000e-02</td>
<td>0.000e+00</td>
<td>0.000e+00</td>
</tr>
<tr>
<td>1.000e-01</td>
<td>9.949e-03</td>
<td>0.995</td>
</tr>
<tr>
<td>1.500e-01</td>
<td>0.000e+00</td>
<td>0.000e+00</td>
</tr>
<tr>
<td>2.000e-01</td>
<td>0.000e+00</td>
<td>0.000e+00</td>
</tr>
<tr>
<td>3.000e-01</td>
<td>4.000e-02</td>
<td>0.490</td>
</tr>
<tr>
<td>4.000e-01</td>
<td>8.500e-02</td>
<td>0.344</td>
</tr>
<tr>
<td>5.000e-01</td>
<td>6.000e-02</td>
<td>0.339</td>
</tr>
<tr>
<td>6.000e-01</td>
<td>1.700e-01</td>
<td>0.273</td>
</tr>
<tr>
<td>8.000e-01</td>
<td>7.500e-02</td>
<td>0.332</td>
</tr>
<tr>
<td>1.000e-00</td>
<td>5.500e-02</td>
<td>0.404</td>
</tr>
<tr>
<td>1.500e+00</td>
<td>3.500e-02</td>
<td>0.404</td>
</tr>
<tr>
<td>2.000e+00</td>
<td>4.500e-02</td>
<td>0.447</td>
</tr>
<tr>
<td>3.000e+00</td>
<td>6.500e-02</td>
<td>0.355</td>
</tr>
<tr>
<td>4.000e+00</td>
<td>8.000e-02</td>
<td>0.327</td>
</tr>
<tr>
<td>5.000e+00</td>
<td>6.500e-02</td>
<td>0.355</td>
</tr>
<tr>
<td>6.000e+00</td>
<td>0.000e+00</td>
<td>0.000e+00</td>
</tr>
<tr>
<td>8.000e+00</td>
<td>0.000e+00</td>
<td>0.000e+00</td>
</tr>
<tr>
<td>1.000e+01</td>
<td>0.000e+00</td>
<td>0.000e+00</td>
</tr>
<tr>
<td>1.500e+01</td>
<td>0.000e+00</td>
<td>0.000e+00</td>
</tr>
<tr>
<td>2.000e+01</td>
<td>0.000e+00</td>
<td>0.000e+00</td>
</tr>
</tbody>
</table>

TOTAL 8.049e-01 0.0589

E TRANS REL ERROR

1.000e-03 0.000e+00 0.000e+00
1.500e-03 0.000e+00 0.000e+00
2.000e-03 0.000e+00 0.000e+00
3.000e-03 0.000e+00 0.000e+00
4.000e-03 0.000e+00 0.000e+00
5.000e-03 0.000e+00 0.000e+00
6.000e-03 0.000e+00 0.000e+00
8.000e-03 0.000e+00 0.000e+00
1.000e-02 0.000e+00 0.000e+00
1.500e-02 0.000e+00 0.000e+00
2.000e-02 0.000e+00 0.000e+00
3.000e-02 0.000e+00 0.000e+00
4.000e-02 0.000e+00 0.000e+00
5.000e-02 0.000e+00 0.000e+00
6.000e-02 0.000e+00 0.000e+00
8.000e-02 0.000e+00 0.000e+00
1.000e-01 0.000e+00 0.000e+00
1.500e-01 0.000e+00 0.000e+00
2.000e-01 0.000e+00 0.000e+00
3.000e-01 0.000e+00 0.000e+00
4.000e-01 0.000e+00 0.000e+00
5.000e-01 0.000e+00 0.000e+00

F-3
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4.000e-01</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>8.000e-01</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>1.000e+00</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>1.500e+00</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>2.000e+00</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>3.000e+00</td>
<td>5.000e-03</td>
<td>0.995</td>
</tr>
<tr>
<td>4.000e+00</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>5.000e+00</td>
<td>1.000e-02</td>
<td>0.700</td>
</tr>
<tr>
<td>6.000e+00</td>
<td>2.500e-01</td>
<td>0.143</td>
</tr>
<tr>
<td>8.000e+00</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>1.000e+01</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>1.500e+01</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>2.000e+01</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

TOTAL 2.650e-01 0.1398

**ABSORB** = 5.101e-05  **CUTOFF** = 0
**TRACKS CREATED BY SPLITT** = 45 **TRACKS LOST TO ROULETTE** = 0
**WEIGHT CREATED BY ROULETTE** = 5.0000e-03 **WEIGHT LOST TO ROULETTE** = 0.0
**TRACKS LOST TO WEIGHT CUTOFF** = 0
**WEIGHT CREATED BY WEIGHT CUTOFF** = 0.0 **WEIGHT LOST TO WEIGHT CUTOFF** = 0.0

TOTAL TIME = 0.0 SECONDS