Nuclear Mathematical and Computational Sciences: A Century in Review, A Century Anew
Gatlinburg, Tennessee, April 6-11, 2003, on CD-ROM, American Nuclear Society, LaGrange Park, IL (2003)

PARALLELIZATION OF THE PENELOPE MONTE CARLO PARTICLE TRANSPORT SIMULATION PACKAGE

R. B. Cruise and R. W. Sheppard
University Information Technology Services
Indiana University
2711 E. 10^{th} Street, Bloomington, IN 47408
rcruise@indiana.edu; rsheppar@indiana.edu

V. P. Moskvin
Department of Radiation Oncology
Indiana University School of Medicine
535 Barnhill Drive, RT-041, Indianapolis, IN 46202
vmoskvin@iupui.edu

ABSTRACT

We have parallelized the PENELOPE Monte Carlo particle transport simulation package [1]. The motivation is to increase efficiency of Monte Carlo simulations for medical applications. Our parallelization is based on the standard MPI message passing interface. The parallel code is especially suitable for a distributed memory environment, and has been run on up to 256 processors on the Indiana University IBM Teraflop SP. Parallel speedup is nearly linear. Fortran 90 features have been incorporated in the parallelized PENELOPE code. The code utilizes the parallel random number generator (p.r.n.g.) developed by the MILC lattice-QCD collaboration [2]. We have tested the parallelized PENELOPE in an application of calibration dosimetry of the Leksell Gamma Knife ® [3]. The parallel results of the test correspond with the serial results, which in turn have been empirically verified in [3]. The parallelized PENELOPE Fortran 90 code, along with a simple example main program, will be available from the Radiation Safety Information Computational Center (RSICC) at Oak Ridge National Laboratory [4] upon completion of benchmark testing.

Key Words: Monte Carlo simulation, MPI, parallel, PENELOPE, radiation transport

1. INTRODUCTION

The use of Monte Carlo (MC) methods for dose computation in radiation oncology applications has been limited by inadequate performance, even with continuing advances in computer architecture and clock speed. Clinical radiotherapy treatment planning systems should provide dose distributions with accuracy of 2–3% of the dose maximum, within an acceptable time limit (less than 5 minutes). Though most MC programs are quite fast for simulating dose deposition in homogeneous media and simple geometries, radiotherapy applications involve numerous variations of material and density over small distances. Patient geometry is usually simulated by a map of densities over a large number of parallelepipeds (voxels). The voxel size ranges from 0.5 to 5.0 mm, and the number of voxels may be $512 \times 512 \times 200$, which is partitioned into 200 512×512 slices obtained from CT scans. Currently, Monte Carlo simulation of the absorbed dose for such large-scale problems requires high-performance computational resources, including parallel programming and computers.

The first parallelization of PENELOPE was implemented by J. Sempau et al. [5]. The code was run on eight processors of an SGI Origin 2000, a shared-memory parallel machine located at the Centre de Supercomputaci ó de Catalunya (CESCA). The parallelization was carried out with the aid of the MPI distributed-memory message passing library, version 1.2, which is discussed below. The authors used the same random number generator (r.n.g.) employed in the serial version of PENELOPE. This r.n.g. combines two different pseudo-random sequences according to the scheme proposed by l'Ecuyer; see [1] for a detailed description. It is a congruential type with period of order 10^{18}. The authors assume one may safely use this approach provided different seeds are introduced in the various processors. The lack of correlation is reasonably guaranteed by the fact that the simulation of each particle involves a random number of calls to the r.n.g. The authors show that results obtained in these studies corroborate the validity of this assumption.

The primary challenge of this work was to ensure the validity of the implementation of the parallel random number generator (p.r.n.g.). Our choice of p.r.n.g. was dictated by this need for validation and is discussed in Section 2.1. A typical PENELOPE run generates 10^8 showers with each averaging 500 calls to the random number generator. Errors in the implementation could easily be hidden by the mass of data. The validity was verified in two ways, first by examining the step-by-step evolution of the p.r.n.g. using a parallel debugger, and second by comparing final results with those of the serial PENELOPE program and with experimental results. Further tests of the statistical validity of the p.r.n.g. are forthcoming in a separate article.

2. DETAILED STRUCTURE OF PARALLELIZED PENELOPE

The generation of the trajectory ensemble is an "embarrassingly" parallelizable computational task. Each parallel processor may read the same geometry and material files specifying the target, and each may generate its own set of particle trajectories. No inter-processor communication is necessary while the trajectory ensemble is being developed. Each processor accumulates relevant raw data from its own trajectory set. Once all processors have completed, a single processor collects the raw data from the others, sums the data into grand totals, and computes ensemble averages and variances. This same processor produces the output, and the program terminates on all processors.

The computational model employed in the parallelization of PENELOPE is the very convenient SPMD (single program, multiple data) model. In this model the distribution of work among the parallel processors is effected not by each performing a unique task or program. Rather, each processor performs a common task, but on a unique input data set. Only one program exists. Each processor executes this same program. However, each processor uses a unique sequence of random numbers, the unique input data set, from which to build its own unique set of trajectories. The critical aspect in the parallelization of PENELOPE, therefore, is the replacement of PENELOPE's original serial random number generator `RAND` with a proven parallel random number generator. This parallel random number generator must be capable of producing a sequence of independent, uniformly distributed random numbers on each processor such that the processors' sequences are mutually uncorrelated.

The current standard API (application programming interface) for implementing a SPMD program is MPI, the Message Passing Interface first released in June 1994 [6]. Our parallelization of PENELOPE is accomplished with the IBM MPI version 3.2. MPI provides the parallel programmer a library of convenient subroutines for managing interprocessor communication. MPI also assigns each processor a unique identification number. This ID may serve as initialization for a parallel random number generator,

resulting in a unique random number sequence for each processor.

2.1. Parallelization of the PENELOPE Subroutines

The key to parallelizing the PENELOPE Fortran 77 subroutines is the replacement of subroutine RAND, the serial random number generator supplied with the PENELOPE package, with a proven parallel random number generator (p.r.n.g.). We have chosen the highly successful p.r.n.g. developed and used by the MILC lattice-QCD collaboration [2]. We considered employing a SPRNG p.r.n.g. [7] but opted instead for the MILC p.r.n.g. in anticipation of our need to perform detailed verification of its implementation. The advantage of the MILC p.r.n.g. is its ease of implementation and easy-to-understand open-source code. Each processor needs only to instantiate a structure with its processor rank, then pass this structure as the argument in calls to the p.r.n.g. It was easy to track the detailed evolution of these structures using a parallel debugger, enabling us to verify the implementation of the MILC p.r.n.g. For instance, this step-by-step tracking revealed the MILC p.r.n.g. has a nonzero probability of returning zero, which is unacceptable to some of the Penelope subroutines, and we were able to easily discover and correct for the problem. We have, on the other hand, no evidence that actual use of the MILC p.r.n.g. is better than some other parallel generator. However, extensive testing of the MILC p.r.n.g. has shown its statistical performance is as good as the original r.n.g. of the serial Penelope distribution. Details of these statistical tests are forthcoming in a separate article.

The MILC p.r.n.g. employs a Marsaglia-like 127 bit feedback shift register. The advantages of a shift register approach compared to a linear congruential or lagged Fibonacci approach are (i) the speed of logical operations compared to arithmetical operations, and (ii) the pseudo-random characteristics of bit-mixing compared to purely arithmetical operations [8]. The period of the 127 bit feedback shift register is $2^{127} - 1 \simeq 10^{38}$; if used serially a gigaflop machine would not repeat its random sequence until 10^{23} years have elapsed. Going one step further, the combination of a shift register with another algorithmically different generator is a very effective means of suppressing any residual correlations in the shift register [8]. The MILC p.r.n.g. combines its 127 bit feedback shift register with a 32 bit integer congruence generator. Parallelization is effected in the MILC p.r.n.g. by each processor using a different multiplier in the congruence generator and a different initial state in the shift register, so that each processor's random sequence is uncorrelated with the others' sequences. The input to this unique initialization is the individual MPI processor ID mentioned above. The MILC p.r.n.g. has served the eight-member MILC collaboration for more than a decade in thousands of Monte Carlo lattice-QCD computations, resulting in well over a hundred publications verifying the theoretical predictions of QCD. Throughout this research programme no deficiencies in the MILC p.r.n.g. have been detected.

However, it is known that a r.n.g. performing well in one application may when employed in another application introduce artifacts into the latter's simulation results. Therefore, according to Ferrenberg et al. [9], "a specific algorithm must be tested together with the random number generator being used regardless of the tests which the generator passed." We are currently testing the MILC p.r.n.g. alone and together with parallelized PENELOPE. The test system is based on the classical Sobol scheme [10]. With various seeds the p.r.n.g. is tested alone and together with parallelized PENELOPE based on Pearson's chi-squared criteria. Tests include:

i) one-dimensional uniformity test (hypothesis H1)

ii) frequency and pair test (bivariate uniformity test H2)

iii) serial and poker test (H3)

iv) gap test (H4).

Tests (i) and (ii) are classical r.n.g. tests. However, they are not sufficient. Tests (iii) and (iv) are performed on sampled random number subsets used in actual PENELOPE simulations. The subset length for tests (ii) and (iii) will be typical for generating one trajectory. Test (iv) applies to both a subset used for generating a single trajectory, and to the entire set used in the complete simulation. At present the stand-alone tests of the MILC p.r.n.g. are completed. The results are no worse than for l'Ecuyer's generator used in the serial version of PENELOPE. Complete test results will be presented later.

The MILC p.r.n.g. subroutine is named `myrand`. The C code for `myrand` may be obtained from the MILC collaboration web site [2]. The use of the term "myrand" indicates that if I am one of the parallel processors, my calls to function `myrand` return a random number sequence unique to me. We slightly modified the MILC p.r.n.g. to eliminate the possibility of returning zero; several PENELOPE subroutines fail for a random variable of zero.

In the SPMD parallelization model each processor calls the same function `myrand` but with a unique data set passed as the argument. These unique parameters are contained inside a structure. A structure is a user-defined type (derived type) that contains a set of elements that themselves may be of various types. It is a programming device ideal for containing the parameters of the random sequence. Each processor is programmed with such a structure, but the parameter values contained in each processor's structure are unique to that processor.

Fortran 77, the programming language in which the PENELOPE subroutines are written, does not support derived types or structures. Fortran 77 recognizes only implicitly defined types (`INTEGER`, `REAL`, etc), while its only container is the array in which all elements must be of the same implicit type. On the other hand, Fortran 90 does support structures, and a Fortran 90 compiler will compile Fortran 77 code. By compiling PENELOPE with the IBM AIX xlf90 Fortran 90 compiler, we are able to insert structures containing the parameters of the random sequence into the PENELOPE subroutines. The `myrand` function takes a single argument which is this structure. Throughout the PENELOPE subroutine code, calls to `RAND` are replaced with calls to `myrand` with this structure as argument. The derived type of this structure is termed `PRNDATA`. The definition of this type is housed in a Fortran 90 module named `PRNDATA_TYPE`, which may be placed in the file containing the main program:

```
MODULE PRNDATA_TYPE
  TYPE PRNDATA
  SEQUENCE
    LOGICAL*4 R0,R1,R2,R3,R4,R5,R6
    LOGICAL*4 MULTIPLIER,ADDEND,IC_STATE
    REAL*4 SCALE
  END TYPE PRNDATA
END MODULE PRNDATA_TYPE
```

This module is shared among both the main program and all subroutines calling `myrand` by the initial line in the main program or subroutine:

```
USE PRNDATA_TYPE
```

which is followed by the declaration of the structure itself:

```
TYPE (PRNDATA) MYNODE_PRNDATA
```

where MYNODE_PRNDATA is the unique structure containing the parameters of the random sequence for each processor. Recall that in the SPMD parallelization model used here, each processor executes the same program. Each processor reads and executes the above lines of code. What distinguishes one processor from another is the unique initialization of the parameter values in each processor's MYNODE_PRNDATA structure. This initialization must take place in the user's main program, discussed below.

Note that any PENELOPE subroutine calling myrand is modified to take an additional argument which is the MYNODE_PRNDATA structure. For instance, PENELOPE's KNOCK subroutine is modified to:

```
SUBROUTINE KNOCK(DE, ICOL, MYNODE_PRNDATA)
```

2.2. Parallelization of the User's Main Program

The PENELOPE package consists only of subroutines. The user must write the main program which repeatedly calls PENELOPE's subroutines to generate an ensemble of trajectories, then computes statistical averages. For parallelizing main programs the user need have only a rudimentary knowledge of parallel programming and MPI.

In distributed parallel programming, each processor is assigned a unique identification number called MYRANK. The first task of the parallelized main program is to initialize the MPI API and assign each processor its unique ID:

```
CALL MPI_INIT(MPIERR)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, MYRANK, MPIERR)
```

The second task is for each processor to obtain the total number of processors NPROCS working in parallel:

```
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, NPROCS, MPIERR)
```

Each processor's MYNODE_PRNDATA structure is initialized with MYRANK via a call to the MILC p.r.n.g.'s initializing C-code function:

```
CALL initialize_prn(MYNODE_PRNDATA, %VAL(SEED), %VAL(MYRANK))
```

Calls to the p.r.n.g. from within the main program are accomplished simply with myrand(MYNODE_PRNDATA), and as mentioned above, calls to PENELOPE subroutines that require random number generation must include the MYNODE_PRNDATA structure, such as JUMP(DSMAX(MAT),DS,MYNODE_PRNDATA).

The main program divides the workload among the parallel processors. The user is free to decide this partitioning of trajectories. We have chosen an equal partitioning. If NTOT is the total number of primary trajectories to be generated, and MYNTOT is the number for each processor, then the following code performs an equal partitioning:

```
MYNTOT = NTOT/NPROCS
IF(MYRANK.LT.(NTOT - MYNTOT * NPROCS)) MYNTOT = MYNTOT + 1
```

The shower simulation now loops over a counter N ranging from 1 to MYNTOT. The PENELOPE timer must be replaced with a parallel wall-clock timer. The MPI wall-clock timer is convenient. At the beginning of the shower simulation loop code:

```
STARTTIME = MPI_WTIME()
```

and later at the end of the shower simulation code:

```
ELAPSEDTIME = MPI_WTIME() − STARTTIME
```

A processor's completion of its portion of trajectories is tested by:

```
IF(N.LT.MYNTOT .AND. ELAPSEDTIME.LT.ITIME) GO TO 101
```

where ITIME is the maximum wall-clock time allowed for the simulation, and where line 101 increments the shower counter N=N+1 and initiates the generation of another primary trajectory.

During the simulation each processor tracks its own trajectory data. For every datum of a serial main program, the corresponding parallel main program requires two data. For instance, the parallel main requires not only the usual declaration for the total number of secondary particles:

```
DIMENSION SEC(3,3)
```

but also a declaration for the number of secondary particles generated by each processor:

```
DIMENSION MYSEC(3,3)
```

At the conclusion of the shower simulations on all the parallel processors, the processor with MYRANK equaling 0 collects and sums the processors' values for MYSEC into the grand total SEC. This collection and amalgamation is very easily programmed with one single MPI function call:

```
CALL MPI_REDUCE(MYSEC,SEC,9,MPI_DOUBLE_PRECISION,MPI_SUM,0,MPI_COMM_WORLD,MPIERR)
```

At this point all processors but processor 0 are done. Processor 0 computes the desired statistics and variances, and produces the output. This is easily programmed by nesting all this activity inside the if statement:

```
IF(MYRANK.EQ.0) THEN
```

Finally, all processors terminate with the conclusion of the main program:

```
CALL MPI_FINALIZE(MPIERR)
STOP
END
```

See Figure 1 for a simplified flow chart of parallelized PENELOPE. It is very important to recognize that each parallel processor executes this same flow chart. See [6] for a detailed description of the above MPI function calls. The parallelized PENELOPE Fortran 90 code, along with a simple example main program, will be available from the Radiation Safety Information Computational Center (RSICC) at Oak Ridge National Laboratory [4] upon completion of benchmark testing.

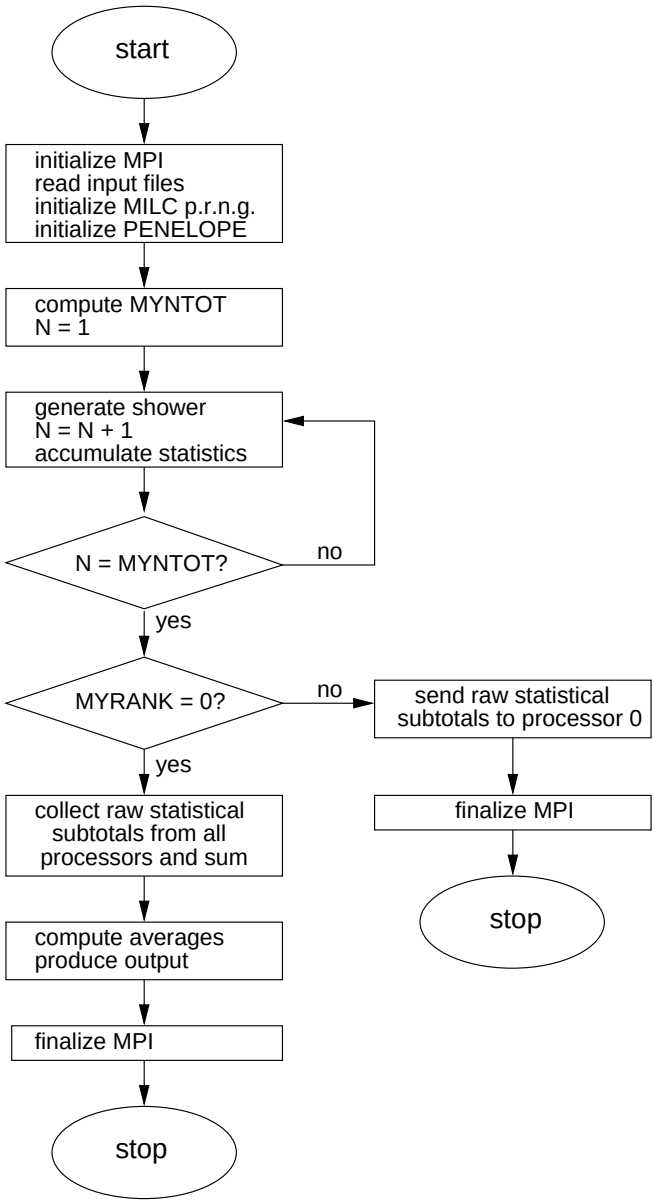

Figure 1. Simplified Flow Chart for Parallelized PENELOPE

3. APPLICATION OF PARALLELIZED PENELOPE

Our motivation in parallelizing PENELOPE is to increase efficiency of Monte Carlo simulations and reduce run-time for medical physics applications. We have applied our parallelized PENELOPE to the simulation of photon beams from the Leksell Gamma Knife®, a precision method for treating intracranial lesions. The Gamma Knife® concentrates radiation from 201 ^{60}Co sources partially arrayed about the exterior of a spherical helmet. Each source includes a primary and final collimator system creating beam diameters of approximately 4, 8, 14, and 18 mm at the common isocenter of the beams. Individual sources may be plugged to create an optimal configuration. Modeling of the Gamma Knife® using the serial version of PENELOPE is detailed in [3]. The parallelized main program used in testing parallelized PENELOPE is based on this same model.

Table I. Parallel Performance for Gamma Knife® Simulation with 10^8 Showers

Number of Processors (N)	Wall-Clock Time T (sec)	Speedup (T_1/T_N)	Efficiency (%)
1	65109	—	—
4	16378	3.98	99.5
8	8162	7.98	99.8
16	4078	15.97	99.8
32	2059	31.62	98.8
64	1028	63.34	99.0
128	518	125.69	98.2
256	271	240.25	93.8

The parallelized PENELOPE code has been compiled and tested on the Aries Complex of the Indiana University IBM Teraflop SP System. The Aries Complex includes eight frames of 4-cpu Power3+ Thin nodes, providing a homogeneous parallel processing environment with a total of 508 processors. The nodes within the Aries Complex are interconnected via a low-latency high-speed (150 megabytes/second) network using crossbar switch technology. Each node runs its own copy of IBM's Unix operating system, AIX 5.1.0. The parallelized PENELOPE code has been compiled and linked with the IBM XL Fortran 90 compiler, version 7.1.1. The C code module containing the MILC p.r.n.g. has been compiled with the IBM C 6.0β compiler. Parallelization is managed by the IBM Parallel Environment 3.2, which includes the IBM MPI libraries. For additional details of the Indiana University IBM Teraflop SP System, including both the Aries and Orion Complexes, see [11].

The run-time test results for 10^8 primary particle trajectories are listed in Table I and plotted in Figure 2. The parallel speedup S_N resulting from N processors is defined by the formula:

$$S_N \quad := \quad \frac{\text{serial run time } \ T_1}{N\text{-processor run time } \ T_N} \tag{1}$$

where times are wall-clock time, not CPU time. The parallel efficiency E_N is defined:

$$E_N \quad := \quad \frac{S_N}{N} \tag{2}$$

The speedup is nearly linear while the parallel efficiency approaches 100% as expected for embarrassingly parallelizable Monte Carlo algorithms. Efficiency decreases a bit for the 256-processor run; communication overhead, as a fraction of run time, starts taking its toll as the number of processors increases while the number of showers per processor decreases.

The parallel code is validated by comparison of output averages to those of the serial code, which in turn is empirically validated in [3]. Because the parallel code uses a different random number generator than the serial code, its output averages are not numerically identical to those of the serial code, but are statistically identical. The parallel code's output averages fall well within the range of estimated statistical error,

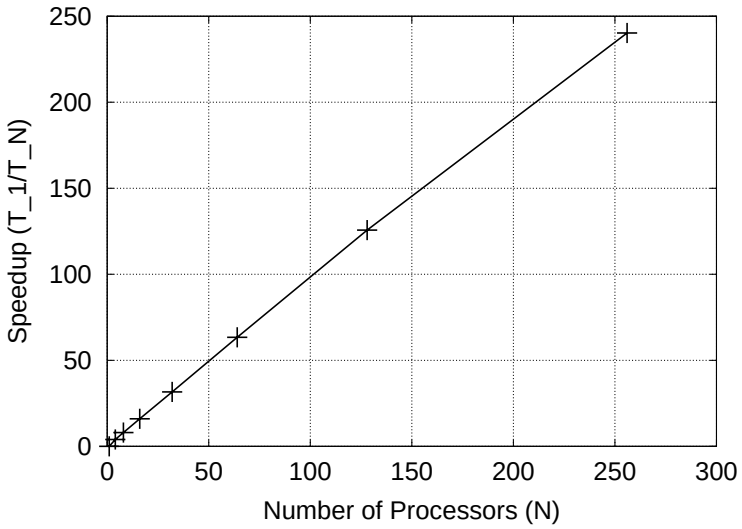

Figure 2. Speedup vs. Number of Processors for Gamma Knife® Simulation with 10^8 Showers

thereby validating the output of the parallel code. Figures 3 and 4 are isodose plots of a spherical polystyrene phantom of radius 16 cm centered at (100,100,100) which corresponds to the isocenter of the Gamma Knife. The plots compare outputs of the serial (dash-dot line) and parallel (solid line) versions of

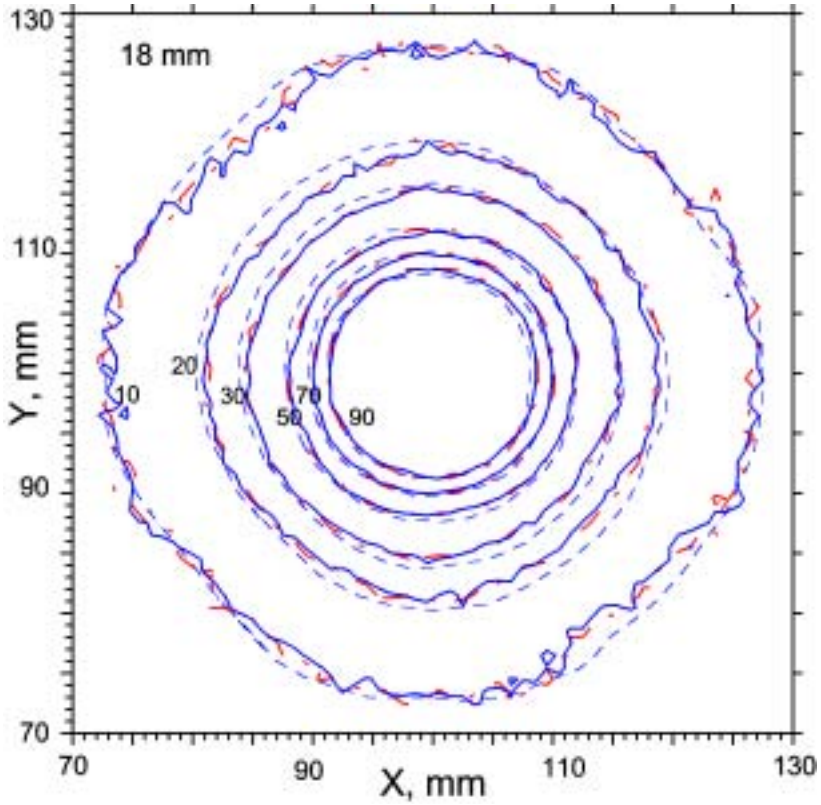

Figure 3. Comparison of dose calculations for Gamma Knife in x, y plane

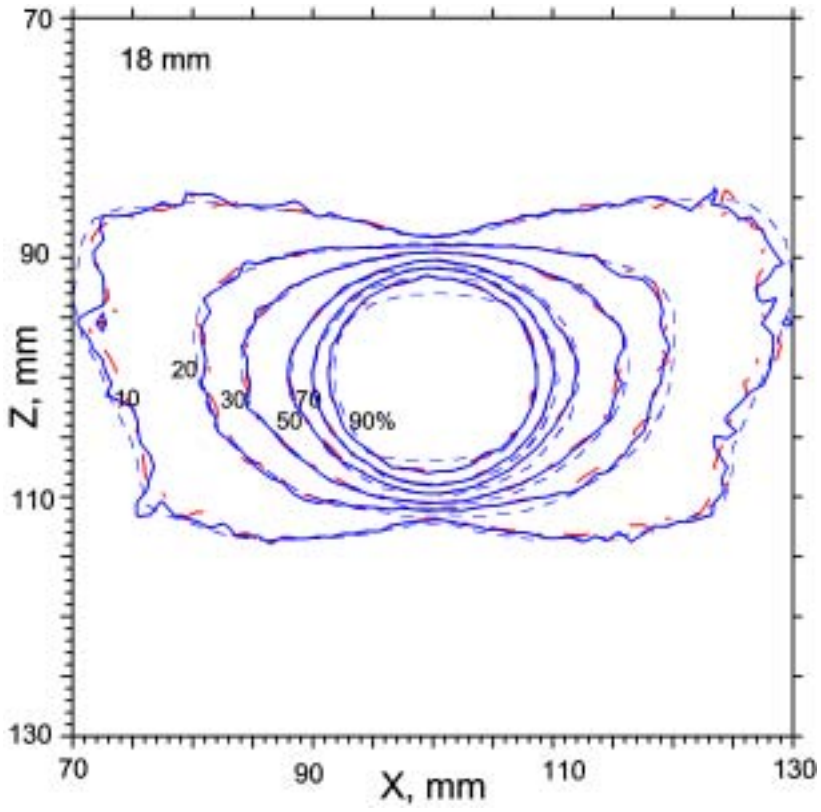

Figure 4. Comparison of dose calculations for Gamma Knife in x, z plane

PENELOPE. The smooth dashed line represents GammaPlan® output. Details of the computational model are presented in [3]. The plots show perfect agreement between parallel and serial versions of PENELOPE. Calculations for parallel and serial cases were done with 10^8 primary photon trajectories corresponding to a statistical error of 3% in dose calculations at the isocenter. The collimator for an 18 mm final beam size was used in the simulation.

4. CONCLUSIONS

Our Fortran 90 parallelization of PENELOPE for execution on distributed-memory machines running MPI exhibits near perfect efficiency. The use of parallelized PENELOPE is completely justified from a resource-utilization standpoint. The output is statistically close to the serial output, which in turn is empirically validated in [3]. The code is presently running on the Aries Complex of the Indiana University IBM Teraflop SP System. However, the Fortran 90 features and MPI function calls in the parallel code are standard, and porting to other parallel Unix platforms should present no difficulties. The parallelized PENELOPE Fortran 90 code, along with a simple example main program, will be available from the Radiation Safety Information Computational Center (RSICC) at Oak Ridge National Laboratory [4] upon completion of benchmark testing.

ACKNOWLEDGEMENTS

This research has been funded in part by the Shared University Research Grants [SUR] from International Business Machines Inc., and the Indiana Genomics Initiative [INGEN]. The Indiana Genomics Initiative of Indiana University is supported in part by Lilly Endowment Inc. Monte Carlo simulation of the Leksell Gamma Knife® is an internal project of the Department of Radiation Oncology, Indiana University.

REFERENCES

[1] F. Salvat, J. M. Fernández-Varea, J. Baró, J. Sempau, "PENELOPE, an algorithm and computer code for Monte Carlo simulation of electron-photon showers," *Informes Técnicos Ciemat*, **799**, CIEMAT, Madrid, Spain 1996.

[2] "The MIMD Lattice Computation (MILC) Collaboration," C. Bernard, T. DeGrand, C. DeTar, S. Gottlieb, U. M. Heller, J. Hetrick, R. L. Sugar, D. Toussaint; the MILC code is available at `http://www.physics.utah.edu/ detar/milc/` (2002).

[3] V. Moskvin, C. DesRosiers, L. Papiez, R. Timmerman, M. Randall, P. DesRosiers, "Monte Carlo simulation of the Leksell Gamma Knife®: I. Source modelling and calculations in homogeneous media," *Physics in Medicine and Biology*, **47**, pp. 1995-2011 (2002).

[4] "Radiation Safety Information Computational Center," Oak Ridge National Laboratory, `http://epicws.epm.ornl.gov/rsic.html` (2002).

[5] J. Sempau, A. Sanchez-Reyes, F. Salvat, H. Oulad ben Tahar, S. B. Jiang, and J. M. Fernandez-Varea, "Monte Carlo simulation of electron beams from an accelerator head using PENELOPE," *Phys. Med. Biol.*, **46**, pp. 1163–1186 (2001).

[6] W. Gropp, E. Lusk, A. Skjellum, *Using MPI*, 2nd ed., Massachusetts Institute of Technology (1999).

[7] M. Mascagni, "SPRNG: A Scalable Library for Pseudorandom Number Generation," *Proceedings of the 4th International Conference on Numerical Methods and Applications*, Sofia, Bulgaria 19 - 23 August 1998, O. P. Iliev, M. S. Kaschiev, S. D. Margenov, B. H. Sendov, P. S. Vassilevski, editors, World Scientific, 1999.

[8] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, *Numerical Recipes in Fortran 90 – The Art of Parallel Scientific Computing*, 2nd ed., Cambridge University Press (1996).

[9] A. M. Ferrenberg, D. P. Landau, and Y. J. Wong, "Monte Carlo simulations: hidden errors from 'good' random number generators," *Phys. Rev. Letters*, **69**, pp. 3382–3384 (1992).

[10] I. M. Sobol, *Numerical Monte Carlo Methods*, NAUKA, Moscow (1973) (in Russian).

[11] "The UITS Research SP System," Indiana University, `http://sp-www.iu.edu/` (2002).