Implementation of the DPM Monte Carlo code on a parallel architecture for treatment planning applications

Neelam Tyagi[a)]
Departments of Nuclear Engineering and Radiation Oncology, The University of Michigan, Ann Arbor, Michigan 48109-2104

Abhijit Bose
Center for Advanced Computing, The University of Michigan, Ann Arbor, Michigan 48109-2104

Indrin J. Chetty
Department of Radiation Oncology, The University of Michigan, Ann Arbor, Michigan 48109-0010

(Received 12 February 2004; revised 6 June 2004; accepted for publication 8 July 2004; published 27 August 2004)

We have parallelized the Dose Planning Method (DPM), a Monte Carlo code optimized for radiotherapy class problems, on distributed-memory processor architectures using the Message Passing Interface (MPI). Parallelization has been investigated on a variety of parallel computing architectures at the University of Michigan-Center for Advanced Computing, with respect to efficiency and speedup as a function of the number of processors. We have integrated the parallel pseudo random number generator from the Scalable Parallel Pseudo-Random Number Generator (SPRNG) library to run with the parallel DPM. The Intel cluster consisting of 800 MHz Intel Pentium III processor shows an almost linear speedup up to 32 processors for simulating 1×10^8 or more particles. The speedup results are nearly linear on an Athlon cluster (up to 24 processors based on availability) which consists of 1.8 GHz+ Advanced Micro Devices (AMD) Athlon processors on increasing the problem size up to 8×10^8 histories. For a smaller number of histories (1×10^8) the reduction of efficiency with the Athlon cluster (down to 83.9% with 24 processors) occurs because the processing time required to simulate 1×10^8 histories is less than the time associated with interprocessor communication. A similar trend was seen with the Opteron Cluster (consisting of 1400 MHz, 64-bit AMD Opteron processors) on increasing the problem size. Because of the 64-bit architecture Opteron processors are capable of storing and processing instructions at a faster rate and hence are faster as compared to the 32-bit Athlon processors. We have validated our implementation with an in-phantom dose calculation study using a parallel pencil monoenergetic electron beam of 20 MeV energy. The phantom consists of layers of water, lung, bone, aluminum, and titanium. The agreement in the central axis depth dose curves and profiles at different depths shows that the serial and parallel codes are equivalent in accuracy. © *2004 American Association of Physicists in Medicine.* [DOI: 10.1118/1.1786691]

Key words: Monte Carlo, MPI, SPRNG, parallel computing

I. INTRODUCTION

The increased accuracy of Monte Carlo calculations in radiotherapy treatment planning applications is accompanied by a significant increase in computational burden—this still remains perhaps one of the biggest drawbacks of the routine use of Monte Carlo in a clinical setting. Since the Monte Carlo method is inherently parallel due to the independent nature of particle transport the use of parallel processing for Monte Carlo simulation offers an attractive approach toward improving the overall computational time. This increase in performance coupled with the use of efficient and accurate codes that have been optimized for radiotherapy, such as, DPM,[1] VMC,[2] etc. may finally make feasible the use of Monte Carlo for routine clinical treatment planning.

Parallelization of Monte Carlo codes is a straight forward approach and is based on distributing the job (number of histories) among different processors which work independent of each other in parallel and their final result is accumulated. Several existing radiotherapy Monte Carlo[3,4] codes have been parallelized using different approaches such as Parallel Virtual Machine (PVM)[5] or Message Passing Interface (MPI)[6] or a Linux shell script. PVM provides for message passing between homogeneous or heterogeneous computers and has a collection of library routines that the user can employ with C or FORTRAN programs. MPI is a standards-based message passing library for a set of processing elements, typically with distributed memory. It is also one of the most popular interfaces for parallelizing existing serial applications.

In a perfect world, the reduction in computation time would be directly linear with the number of processors. However, Amdahl's law[7] prevents us from achieving perfect parallelism. The serial part of the code limits the speedup and efficiency of the overall code. The overhead associated with communication and synchronization leads to further degradation in speedup and efficiency. One can define speedup, S_N, as the ratio of execution time on a single pro-

cessor (T_1) to the execution time using N (T_N) processors, i.e., $S_N = T_1/T_N$. Since most scientific codes have a serial and a parallel portion, the definition of speedup can also be written as

$$S_N = \frac{1}{f_s + \frac{f_p}{N}}, \quad (1)$$

where f_s = serial fraction on one processor and f_p = parallel fraction of the code on one processor and $f_s + f_p = 1$. Equation (1) describes Amdahl's law and also implies that the maximum achievable speedup (obtained as the number of processors, $N \to \infty$) is $1/f_s$, i.e., speedup is limited by the portion of the code that is serial. The serial portion has a strong limiting effect on the speedup but can often be minimized by increasing the problem size, which tends to reduce the serial fraction for many applications. One can also define the efficiency of the parallel code with N processors as the ratio of the speedup to the number of processors $\varepsilon_N = S_N/N$. Since $S_N \leqslant N$, we have $\varepsilon_N \leqslant 1$. Perfect speedup is achieved when $S_N = N$, and $\varepsilon_N = 1$. In practice, superlinear speedup may be achieved whenever the speedup exceeds the number of processors. The most common causes for superlinear speedup are cache effects and randomized algorithms. As the number of processors increases for a fixed problem size, the smaller problem size on each processor results in higher cache hits as compared to large cache misses for the single-processor case. Therefore, certain applications and problem sizes may exhibit a super linear speedup up to a certain number of processors. If the communication costs grow with increasing number of processors, eventually the gain due to cache hits is offset by the increased communication.

This study describes the implementation of the DPM Monte Carlo code on a variety of parallel computing architectures, and investigates the sources of degradation in efficiency and speedup in the parallel version of the code.

II. IMPLEMENTATION OF DPM ON A PARALLEL ARCHITECTURE

A. Parallelization

The DPM code was parallelized on a Linux cluster using the Message Passing Interface (MPI) for interprocessor communication. Temporary buffers are assigned within DPM and are used by the master and slave processors to store dose and error values. Eight basic standard MPI calls were used for parallelization. The basic functions of the master processor include reading the initial datasets (such as number of histories, electron and photon energy cut-offs, the voxel geometry file, cross-section files and region of interest for dose output) from the DPM input file and broadcasting this information to the other processors. At the end of the simulation, the master processor collects and combines all the dose values calculated by the slaves. The slave processors read the broadcasted information, write dose values in temporary buffers and send the calculated data when requested by the master processor. A detailed description of the parallelization is shown in the flowchart (Fig. 1). The original code is written in FORTRAN 77 and uses GNU compilers on an UNIX environment. The parallel MPI version is compiled with mpif77 and mpiCC.

B. Parallel architectures under study

The speedup and efficiency were tested on three different Linux clusters of different processor architectures:

(a) An Intel cluster consisting of 50 nodes (2 CPU's/node), 800 MHz Intel Pentium III processors. Each node shared a common memory pool of 1 GB RAM.

(b) An Athlon cluster consisting of 50, 1.8 GHz+ AMD Athlon nodes (2 CPU's/node). The dual processors on each node are configured with 2 GB RAM and therefore, can operate as a SMP (Symmetric Multiprocessors) node. The Athlon MP has three out-of-order, superscalar and fully-pipelined floating point execution units and is well-suited for scientific and engineering computations. Moreover, the system bus with a peak rate of 2.1 GB/s provides high bandwidth for data-intensive applications. The nodes in the Athlon cluster are interconnected via a Myrinet[8] 2000 switch that provides sustained one-way data rate of 248 MB/s, and a low latency (6.3 μs).

(c) An Opteron cluster consisting of 100 nodes (2 CPU's/node) of 64-bit AMD Opterons with a CPU speed of 1400 MHz. Of these, a total of 36 nodes share a memory pool of 2 GB/node, 32 nodes share 4 GB/node and another 32 nodes share 6GB/node. The Opteron processor provides up to 6.4 GB/s of memory bandwidth per processor reducing the memory latency and I/O bottlenecks. In addition, the AMD Hyper Transport technology allows up to three coherent links, or 19.2 GB/s of peak bandwidth per processor. A significant feature of the Opteron processor is its ability to simultaneously execute both 32- and 64-bit binaries natively. The Opteron cluster uses a dedicated Force-10 Gigabit Ethernet switch for interconnection of the nodes. While Gigabit Ethernet provides approximately half the bandwidth of a corresponding Myrinet switch, it can be sufficient for many scientific applications requiring moderate interprocessor communication and taking advantage of overlapping computation with communication.

C. Scalable parallel pseudo random number generator

An important element in history-based parallel algorithms is a reliable parallel random number generator that provides uncorrelated random number streams to each processor. The parallel random number sequence should also satisfy the criteria of an acceptable serial random number stream, i.e., it should be sufficiently uniform, have a large period and have no correlation between the numbers in the sequence. The original serial version of DPM Monte Carlo code uses a 64-

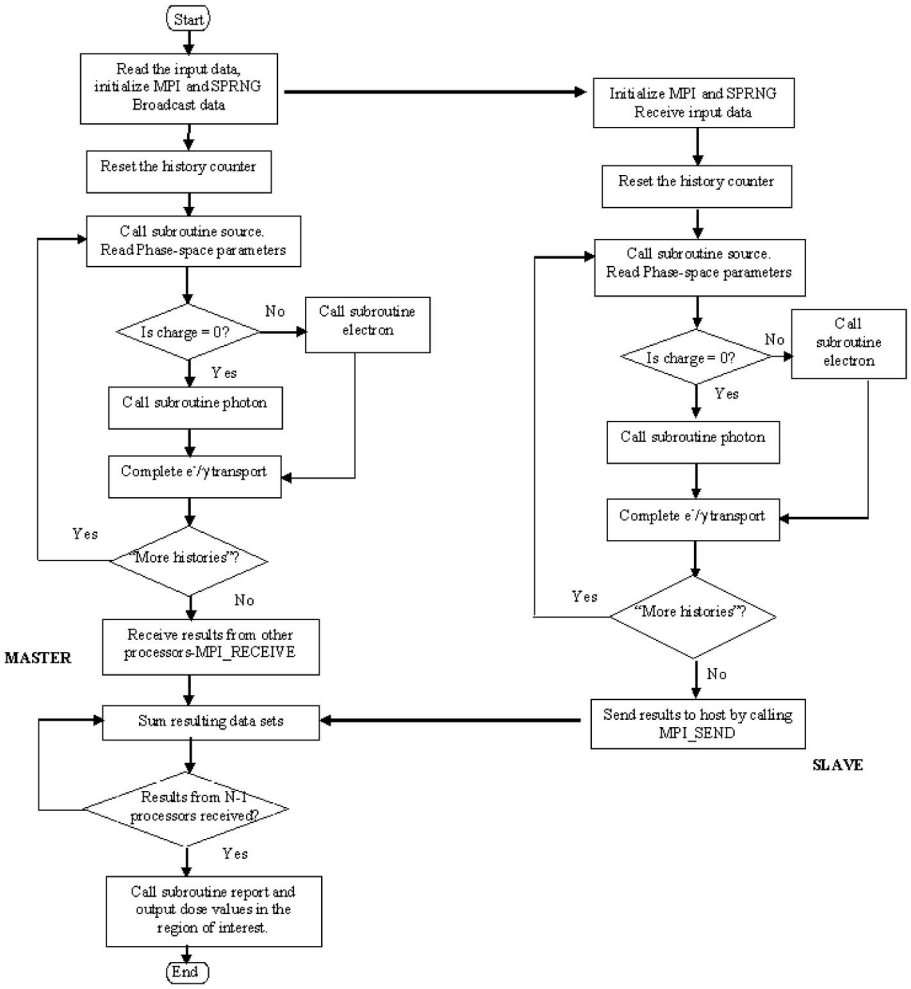

FIG. 1. Flowchart describing the parallelization of the DPM Monte Carlo Code.

bit random number generator from PENELOPE. We have implemented the parallel pseudo random number generator from the SPRNG (Ref. 9) library in parallel DPM. SPRNG is a set of libraries for scalable and portable pseudorandom number generation. It has been ported on a variety of computing platforms and supports MPI. We modified the relevant subroutine within DPM to incorporate SPRNG. SPRNG uses an initial seed which refers to the encoding of the starting state rather than the conventional notion of the starting state of the random number stream. Distinct streams initialized with the same initial seed have different starting states. This makes it convenient to use the same seed for distinct streams and still obtain different initial states. We compiled SPRNG with the gcc/g77 and the PGI (Portland group) version of MPI. It was also tested with LAM-MPI.[10]

III. TIMING RESULTS

Tables I, II, and III show the efficiency and speedup estimates as calculated on the Intel, Athlon, and Opteron clusters. These calculations are based on a simulation of a 6 MV photon beam in a $25.6\times25.6\times30$ cm^3 water phantom with voxel size $0.2\times0.2\times0.2$ cm^3. The time T (min) taken for the simulation is the time required for the in-phantom dose calculation using the DPM code. This time shows an inverse relation to the processor speed of the three Linux clusters. Three sets of calculations were performed for each analysis depending on the availability of processors and an estimate of uncertainty is evaluated. The standard deviation for all calculated times was within 5%. The Intel cluster with the slowest 800 MHz processors, took the largest amount of simulation time. The Athlon cluster being twice as fast as Intel took approximately half the simulation time as is noted in Table II. However in comparing the efficiency ε between Intel and Athlon, we find that Intel shows consistently better efficiency up to 32 processors for simulating 1×10^8

TABLE I. Speedup and efficiency on an Intel cluster (Number of histories $=1\times10^8$).

Nodes	No. of CPUs	T (min)	S_{Intel}	$\varepsilon_{\text{Intel}}$ (%)
1	1	82.9	1.0	100.0
1	2	40.3	2.0	100.0
2	4	21.1	3.9	98.4
4	8	10.5	7.9	99.2
8	16	5.3	15.6	97.5
10	20	4.4	18.6	93.2
16	32	3.0	27.6	86.4

TABLE II. Speedup and efficiency on an Athlon cluster.

Nodes	No. of CPU's	$T(\min)\pm\sigma(\%)$ (Min, Max) Number of histories 10^8	$T(\min)\pm\sigma(\%)$ (Min, Max) Number of histories 8×10^8	S_{AMD} Number of histories 10^8	S_{AMD} Number of histories 8×10^8	$\varepsilon_{AMD}(\%)$ 10^8	$\varepsilon_{AMD}(\%)$ 8×10^8
1	1	30.9±0.7 (30.6, 31.0)	244.4±0.4 (243.7, 245.8)	1.0	1.0	100.0	100.0
1	2	16.0±1.9 (15.8, 16.4)	127.7±0.02 (127.7, 127.8)	1.9	1.9	96.2	95.7
2	4	8.1±1.5 (8.0, 8.3)	65.4±0.7 (65.1, 66.1)	3.8	3.7	95.2	93.5
4	8	4.3±1.8 (4.2, 4.4)	32.8±0.7 (32.4, 32.9)	7.2	7.5	90.7	93.3
8	16	2.2±0.9 (2.2, 2.2)	16.7±0.8 (16.6, 16.9)	14.1	14.6	88.2	91.3
10	20	1.9±4.3 (1.8, 2.0)	12.9±0.2 (12.9, 12.9)	16.4	18.9	82.1	94.8
12	24	1.5±1.3 (1.5, 1.6)	11.2±0.4 (11.2, 11.3)	20.1	21.8	83.9	90.8

particles. This is because, with the Intel cluster, the amount of overhead associated with interprocessor communication and the serial fraction of the code is masked by the limitation in processor speed. On the other hand, the reduction of efficiency with the Athlon cluster (down to 83.9% with 24 processors) occurs because the processing time required to simulate 1×10^8 particles is less than the time associated with interprocessor communication. On simulating 8×10^8 histories with Athlon, we found an improvement in efficiency (90.8% as compared to 83.9% with 1×10^8 particles simulated). This is also consistent with the fact that the fraction of time spent in the sequential part of an algorithm may actually decrease as the problem size increases making the parallel computation more efficient.

From Table III, we find that the Opteron cluster is faster than the Athlon cluster even though its processor speed is slower. This is due to the fact that Opteron, being a 64-bit architecture is capable of storing and processing instructions at a faster rate as compared to the 32-bit Athlon cluster. Moreover, as mentioned earlier, the Opteron processor has a memory bandwidth much higher than the Athlon resulting in faster data movement in and out of cache, as well as the system bus. The time taken to simulate 1×10^8 histories was 1.2 min on 24 processors. However, the efficiency of the calculation was only 87.0%, which is again associated with fact that the amount of CPU time was significantly smaller than the cost attributed to interprocessor communication and the usage of a slower interconnection mechanism than a Myrinet switch. We performed a similar phantom calculation with 8×10^8 histories on the Opteron cluster and again, there was a significant improvement in efficiency and speedup (96.1% on 24 CPU's). The scalability of the Code on the Opteron cluster was also studied up to 64 processors where it shows an efficiency of 92% and hence are ideally suited for large scale applications.

TABLE III. Speedup and efficiency on an Opteron cluster.

Nodes	No. of CPU's	$T(\min)\pm\sigma(\%)$ (Min, Max) Number of histories 10^8	$T(\min)\pm\sigma(\%)$ (Min, Max) Number of histories 8×10^8	$S_{Opteron}$ Number of histories 10^8	$S_{Opteron}$ Number of histories 8×10^8	$\varepsilon_{Opteron}(\%)$ 10^8	$\varepsilon_{Opteron}(\%)$ 8×10^8
1	1	25.8±0.4 (25.6, 25.8)	204.0±0.1 (203.7, 204.2)	1.0	1.0	100.0	100.0
1	2	13.1±0.9 (13.0, 13.2)	102.7±0.3 (102.4, 103.1)	1.97	1.99	98.6	99.4
2	4	6.6±1.3 (6.5, 6.7)	52.7±0.09 (52.6, 52.7)	3.92	3.87	98.1	96.8
4	8	3.4±0.5 (3.3, 3.4)	26.6±2.1 (26.0, 27.0)	7.69	7.67	96.1	95.9
8	16	1.8±0.5 (1.8, 1.8)	13.4±3.5 (13.1, 14.0)	14.67	15.18	91.7	94.8
10	20	1.5±1.7 (1.4, 1.5)	10.5±0.2 (10.5, 10.6)	17.69	19.35	88.5	96.7
12	24	1.2±0.0 (1.2, 1.2)	8.8±0.9 (8.8, 8.9)	20.88	23.07	87.0	96.1

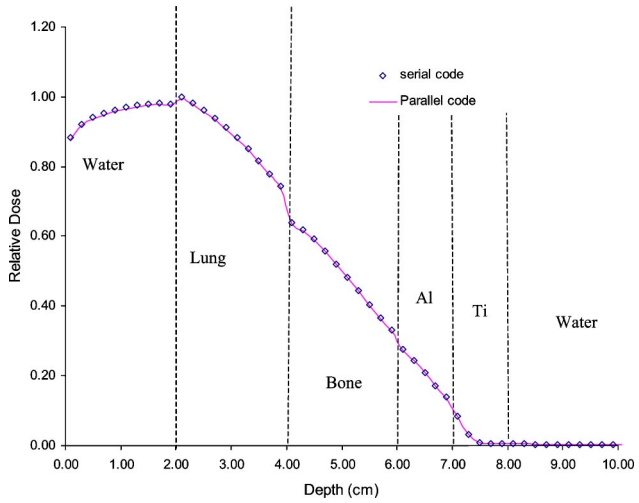

FIG. 2. Central axis depth dose curve comparison between the serial and parallel version of the DPM code in a phantom consisting of layers of water, lung, aluminum, and titanium.

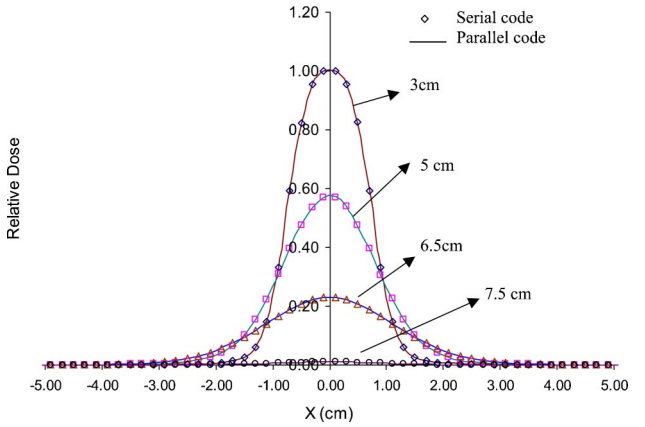

FIG. 3. One-dimensional line profiles at depths of 3 cm, 5 cm, 6.5 cm, and 7.5 cm, respectively, for serial and parallel calculation.

IV. VALIDATION OF PARALLEL IMPLEMENTATION

To evaluate the accuracy and efficiency of the DPM parallel implementation with respect to the serial version, we simulated a phantom consisting of layers of water, lung ($\rho = 0.3$ g/cm^3), bone, aluminum, and titanium. The simulation was carried out on the University of Michigan Radiation Oncology Intel Linux cluster. The cluster consists of 10 dual node Intel Xenon Pentium IV processors with processor speed of 2.4 GHz. We used a parallel pencil electron beam of 20 MeV energy. The phantom size was $25.6\times25.6\times30$ cm^3 with a voxel size of $0.2\times0.2\times0.2$ cm^3. Two sets of calculations were carried out: one with the serial (original) version of the DPM code and the other with the parallel version. 1×10^8 histories were simulated to get an average sigma of less than 0.1%. Figure 2 compares the central axis depth dose curves from the serial and parallel calculations. We see excellent agreement between the serial and parallel codes. Figure 3 shows one-dimensional line profiles in the transverse direction at various depths obtained from the two calculations. The agreement in the profiles illustrates that the serial and parallel codes are equivalent in accuracy. The time taken by the serial code for this simulation (for 1×10^8 particles) was 438 min and the time taken by the parallel code was 46.8 min on 8 processors, and shows that the simulations are far more efficient.

V. CONCLUSION

We have successfully parallelized the DPM Monte Carlo code for distributed-memory processor architectures using the Message Passing Interface (MPI) and tested its efficiency and speedup with respect to number of processors. Results show roughly linear increases with speedup with an increasing number of processors, however, deviations from linearity were noted with varying problem size. Speedup was also found to scale with processor speed. By taking advantage of multiple processors and the continuing increase in processor speeds, it seems likely that parallelization techniques as explored in this study will make feasible the routine use of Monte Carlo for treatment planning in the clinical setting.

ACKNOWLEDGMENTS

The authors would like to thank Professor William R. Martin from the University of Michigan for his help and suggestions. We would also like to thank system administrators at Center for Advanced Computing at University of Michigan for providing the computational resources to complete this study. This work was supported in part by Grant No. NIH P01-CA59827.

[a]Electronic mail: ntyagi@umich.edu

[1] J. Sempau, S. J. Wilderman, and A. F. Bielajew, "DPM, a fast, accurate Monte Carlo code optimized for photon and electron radiotherapy treatment planning dose calculations," Phys. Med. Biol. **45**, 2263–2291 (2000).
[2] I. Kawrakow, M. Fippel, and K. Friedrich, "3D electron dose calculation using a Voxel based Monte Carlo algorithm (VMC)," Med. Phys. **23**, 445–457 (1996).
[3] J. F. Briesmeister, MCNPTM-A general Monte Carlo N Particle transport code, Los Alamos National Laboratory Report No. LA-12625-M, 1997.
[4] R. B. Cruise, R. W. Sheppard, and V. P. Moskvin, "Parallelization of the Penelope Monte Carlo particle transport simulation package," Nuclear Mathematical and Computational Sciences: A century in Review: A century anew, Gatlinburg, Tennessee, April 6–11, 2003.
[5] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Mancheck, and V. Sunderam, *PVM: Parallel Virtual Machine* (MIT Press, Cambridge, 1994b).
[6] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI*, 2nd ed. (Massachusetts Institute of Technology, Cambridge, 1999).
[7] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," Proc. AFIPS Comput. Conference, 1967, Vol. 30, pp. 483–485.
[8] http://www.myri.com/
[9] http://sprng.cs.fsu.edu/
[10] http://www.lam-mpi.org