Parallel Performance Study of Monte Carlo Photon Transport Code on Shared-, Distributed-, and Distributed-Shared-Memory Architectures

Amitava Majumdar
San Diego Supercomputer Center
University of California San Diego
9500 Gilman Drive, La Jolla CA 92093-0505
majumdar@sdsc.edu

Abstract

We have parallelized a Monte Carlo photon transport algorithm. Three different parallel versions of the algorithm were developed. The first version is for the Tera Multi-Threaded Architecture (MTA) and uses Tera specific directives. The second version, which uses MPI library calls, has been implemented on both the CRAY T3E and the 8-way SMP IBM SP with Power3 processors. The third version is a hybrid MPI-OpenMP implementation and is used on the SMP IBM SP. This version uses MPI to communicate between nodes and OpenMP to perform shared memory operations among processors within a node. We explain the three different parallelization approaches and present parallel performance results of these three parallel implementations on three different machines. We observe near perfect speedup for the three versions on the three architectures. The results on the SMP IBM SP suggest that the hybrid MPI-OpenMP programming is suitable for SMP type machines.

1. Introduction

Monte Carlo particle transport is an inherently parallel (or embarrassingly parallel) computational method that has been studied on a number of alternative architecture [1,2,3,4,5,6]. Currently there is interest to simulate enormously large Monte Carlo particle transport problems for neutron and photon transport on teraflop scale machines. Present teraflop scale parallel architectures are multiple node architectures where each node is a Symmetric Multi-processor (SMP). These new architectures encourage a hybrid parallel programming paradigm. In this model computational work load is divided across distributed memory nodes using explicit message passing, and within a node work load is divided across multiple processors using shared memory programming. This does not, however, preclude explicit message passing among multiple processors within a node. Recently there is also interest to explore multithreaded architectures to improve parallel performance of scientific codes.

This paper summarizes recent experiences with adapting a Monte Carlo photon transport code to several latest architecture machines. The code has been adapted to different parallel programming styles, such as purely shared memory (using multithreading), purely message passing (using MPI), and hybrid of message passing and shared memory (using MPI and OpenMP). Parallel versions of the code were implemented on three different parallel architectures. The different parallel architectures targeted are the shared memory Tera MTA, the distributed memory Cray T3E, and the 8-way SMP IBM SP with Power3 processors.

2. Description of Monte Carlo Code

TPHOT, the code used for this parallel performance study, is a time-dependent Monte Carlo photon transport code. TPHOT simulates photon transport within high-density, high-temperature Inertial Confinement Fusion (ICF) plasma in two-dimensional r-z geometry, and includes realistic opacity data. The plasma is divided into zones, each with its own composition, temperature, and density. Each zone is a simple volume of revolution, bounded by at most four surfaces. This geometry allows all particles to be treated simultaneously irrespective of zone.

Photons are sampled uniformly and isotropically within each zone from a Planckian energy spectrum. The energy range is discretized into several energy groups. The number of photons emitted within each zone is a function of material properties of the zone and volume of the zone. The photons that are emitted within each zone and energy group are then followed through the plasma until they are absorbed, escape, or reach

0-7695-0574-0/2000 $10.00 © 2000 IEEE

census, i.e., the end of a time step. Besides absorption, the photons may undergo Thompson scattering.

The overall workload can be divided into four categories, (1) pre-processing work, (2) serial Monte Carlo work, (3) parallel Monte Carlo work, and (4) post-processing work. The pre-processing work includes reading the input data, the serial Monte Carlo work includes preparing geometry and material properties for all the zones, parallel Monte Carlo work includes the actual particle history tracking and accumulating tallies, and the post-processing work includes writing output results. Our measurement of timing does not include the pre-processing and post-processing work.

In TPHOT, the geometry and material properties of zones are constant in time, and time stepping is used only to determine when results are output to a census file. In a more general case, one might model coupled photon transport and hydrodynamics, in which case the geometry and material properties could change from one time step to another. The present simulation includes only the within time step Monte Carlo calculations.

The main part of the particle transport algorithm, i.e. where the parallel Monte Carlo work can be done, has an outer loop over the zones to generate the photons. Inside this loop over zones is a loop over energy groups. In addition, inside the energy group loop is a loop over photons emitted in a particular zone and in a particular energy group. The simulation within a time step continues until the loop over zones has been completed and all of the photons have been emitted and followed. Relevant pseudo-code for these loops follows. In the following pseudo code the variable edep is the energy deposited in each zone and energy group, and it is a function of the Planckian energy spectrum, the material properties, and the volume of each zone.

```
do i = 1, number_of_zones
  do j =  1, number_of_energy_groups
     .
   (update of some tally variables)
    number_of_photons_in_i_and_j = function of (edep)
    do k = 1, number_of_photons_in_i_and_j
call multiple subroutines to track the photon history
(update tallies inside subroutines called from the
innermost loop  over number_of_photons_in_i_and_j)
    end do
  end do
end do
```

Since the photons do not interact with each other, their histories are independent and can be computed simultaneously. For distributed memory machines accumulations of various tallies at the end of the simulation is the only place where communication is necessary. For shared memory machines the tallies are shared variables, which are updated during each history simulation. This allows a embarrassingly parallel implementation, which should exhibit nearly linear speedup, provided that (1) the workload can be balanced by a suitable assignment of photons or zones to processors, and (2) each processor has ready access to the geometry and the material properties required for the tracking of particles assigned to that processor. The second condition implies that the geometry and material properties of the whole domain of interest must fit in the local memory of a processor. In case of distributed memory machines, if the geometry and material properties of the whole domain do not fit in the local memory of a processor then the algorithm, although theoretically embarrassingly parallel, is not embarrassingly parallel in practice. If a particle moves to a zone whose properties are not available in the local memory, then communication would be required among processors while tracking a particle history and the algorithm is no longer embarrassingly parallel.

To obtain reproducible results, each processor should also generate an independent reproducible sequence of random numbers, which are not correlated to another processor's random number sequence. We have used the parallel Linear Congruential random number generator from the NAS 2 Parallel Benchmark [7] suite, which satisfies this condition.

3. Description of parallel machines

3.1. The Tera MTA

The Tera MTA [8, 9] represents a radical departure from traditional vector- or cache-based computers. MTA processors have no data cache or local memory. Instead, they are connected via a network to commodity memory, configured in a shared memory fashion. Randomized memory mapping and high interconnectivity network provide near-uniform access time from any processor to any memory location. Hardware multithreading is used to tolerate high latencies to memory. This latency is typically on the order of 150 clock cycles. The processor can issue an instruction containing a memory reference and two other operations per clock period. The other operations can be floating add and a floating point fused multiply-add. Thus, the theoretical peak speed of a processor is three floating point operations per clock. In practice, no more than two floating point operations per clock have been sustained on realistic computations. Typically, performance is further limited by the network bandwidth, which can return at most one 8 byte (64bit) word to each processor every clock. Expected benefits of the MTA include high processor utilization, near

0-7695-0574-0/2000 $10.00 © 2000 IEEE

linear scalability, and reduced programming effort specially compared to distributed memory machines using explicit message passing.

Currently the largest MTA, the only one other than Tera Company's own machine, is located at the San Diego Supercomputer Center (SDSC). It has eight processors running at 260 MHz and has 8 gigabyte of shared memory.

3.2. The Cray T3E

SDSC's Cray T3E has 272 distributed memory processors of which 260 are available for running dedicated parallel applications. Each processor is a DEC Alpha 21164 chip running at 300 MHz clock speed and has 128 megabytes of memory. The DEC Alpha chips are capable of a theoretical peak speed of 600 MFLOPS. MPI, SHMEM, and other message passing library calls are used to develop parallel programs on this architecture.

3.3. The IBM SP

SDSC recently received the latest Power3 processor based SMP IBM SP nodes. Currently there are 144 SMP nodes with 8 processors per node. Each SMP node has 4 gigabyte of memory shared among its eight Power3 processors running at 222 MHz each. The Power3 processors are capable of executing four floating point operations per cycle. The theoretical peak performance of the Power3 chip is 888 MFLOPS.

The SP nodes allow symmetric multi-processing among the processors within a node and message passing across nodes. MPI, LAPI and other library calls can be used to do message passing across nodes. Within nodes, either OpenMP library calls or pthreads can be used to perform shared memory programming among processors. MPI can also be used to communicate among the processors within a node.

4. Parallelization on the MTA

4.1. Parallelization by zones

Since the computations across zones are independent and the Tera MTA prefers outer-loop parallelization, in our first implementation on the MTA we parallelized the outermost zone loop. The inner loops include many subroutine calls and shared accumulators, and hence the Tera compiler was not able to parallelize the outermost zone loop automatically. Thus an ASSERT PARALLEL directive and several ASSERT LOCAL directives were inserted to force the compiler to parallelize the loop and to identify the local variables respectively. Moreover, each global tally (such as the number of photons escaped, number of photons absorbed, etc.) had to be preceded with an UPDATE directive to insure determinacy. The UPDATE directives perform atomic update on shared variables. The parallel pseudo-code, including the aforementioned directives in bold, is as follows.

```
C$TERA ASSERT PARALLEL
do i = 1, number_of_zones
C$TERA ASSERT LOCAL (various local variables like particle's position, velocity, etc. and local variables required for the random# generator)
  do j =  1, number_of_energy_groups
     .
C$TERA UPDATE directives before each tally statement
    number_of_photons_in_i_and_j = function of (edep)
    do k = 1, number_of_photons_in_i_and_j
call multiple subroutines to track the photon history
(C$TERA UPDATE directives before each tally statement inside subroutines called from the innermost loop over number_of_photons_in_i_and_j)
    end do
  end do
end do
```

As will be seen shortly from Results and discussion section, parallelization done over zones only does not scale well on the MTA. There was insufficient parallelism to hide latency and get good load balance among MTA processors.

4.2. Parallelization by zones and energies

In the second approach, parallelism was increased by collapsing the two outer loops over zones and energies into a single loop. This increased the total number of iterations for the loop. In addition, the order of processing the zones was reversed, since the number of photons in a zone is proportional to its size, and the size of a zone grows with a zone's index value. The resulting pseudo-code is as follows.

```
C$TERA ASSERT PARALLEL
  do ij = number_of_zones*number_of_energy_groups-1,0,-1
C$TERA ASSERT LOCAL (particle's position, velocity, etc. and local variables required for the random# generator)
     .
C$TERA UPDATE before each tally statement
   i = (ij/ number_of_energy_groups) + 1
   j = ij - number_of_energy_groups*(i - 1) + 1
```

0-7695-0574-0/2000 $10.00 © 2000 IEEE

.
number_of_photons_in_ij = function of (edep)
do k = 1, number_of_photons_in_ij
call multiple subroutines to track the photon history
(C$TERA UPDATE before each tally statement inside subroutines called from the innermost loop over number_of_photons_in_ij)
end do
end do

This approach showed near perfect speedup as explained in Results and discussion section.

5. Parallelization on the Cray T3E

The parallel version of TPHOT developed for the Cray T3E uses the MPI library. The parallelization strategy for the T3E is different from that used on the MTA. If NP is the number of T3E processors available, then the parallel pseudo-code for the main computation-intensive part of the code is shown below. The modified part of the pseudo-code is in bold letters.

```
do i = 1, number_of_zones
 do j =  1, number_of_energy_groups
      .
  (update of some tally variables)
  number_of_photons_in_i_and_j =function
of((edep)/NP)
  do k = 1, number_of_photons_in_i_and_j
call multiple subroutines to track the photon history
(update of tally variables inside subroutines called from
the innermost loop over
number_of_photons_in_i_and_j)
  end do
 end do
end do
.
```

Call to multiple MPI_REDUCE(..) to add up all tally variables.

The above modification makes the energy deposited in each zone and each energy group a fraction 1/NP of the total energy for each processor. The net effect is that each of the NP processors simulates 1/NP of the total photons over all the zones. This parallelization strategy effectively parallelizes by distributing 1/NP of the total number of photons to each processor. Since each photon history is independent of any other photon history, it is possible to parallelize this way.

At the end of the simulation one of the NP processors needs to perform multiple MPI reduction operations to add up various global tallies accumulated on each of the NP T3E processors. All these material and mesh structure properties are in COMMON block storage and reside on each processor's local memory. As noted previously one requirement for this parallelization strategy, specifically of concern for distributed-memory machines, is that the geometry and material properties for the whole domain must fit in the memory of each processor.

6. Parallelization on the IBM SP

6.1. Parallelization using MPI on IBM SP

In this approach we used the same parallel version of TPHOT that was used on the T3E i.e. a purely MPI version of the code. To use this version the SMP based SP was treated, from users point of view, as a purely distributed memory machine. The eight processors within a node have access to $1/8^{th}$ fraction of the total memory as long as more than one MPI task is used per node. In current status of the machine, if one processor is used as the only MPI task per node then it has access to only 2 gigabyte of the memory. Message passing was done among the eight processors within a node as well as among processors across nodes using MPI. Another feature of the current machine is that up to 4 MPI tasks on 4 processors can run on a node using the fast switch. To use 8 MPI task per node a slower interconnect switch has to be used.

6.2. Parallelization using MPI-OpenMP on IBM SP

In the hybrid mode of MPI-OpenMP parallelization, TPHOT was first parallelized across the nodes using MPI as described for the T3E parallelization in section 5. One MPI task was assigned to each SMP node. Then within a node OpenMP was used to parallelize the outer most loop over zones similar to the Tera parallelization described in section 4.1. The parallel pseudo code for the hybrid MPI-OpenMP version of TPHOT is as follows, with the modified part of the pseudocode in bold letters.

```
!$OMP PARALLEL DO
!$OMP& PRIVATE(*newly created temporary
tally variables that are tallied in subroutines called
within the inner most loop over
number_of_photons_in_i_and_j)
!$OMP& PRIVATE(particle's position, velocity,
etc. and private variables required for the random#
generator)
!$OMP& reduction(**various tally variables)
do i = 1, number_of_zones
 do j =  1, number_of_energy_groups
    .
 (update some **tally variables)
```

0-7695-0574-0/2000 $10.00 © 2000 IEEE

number_of_photons_in_i_and_j =function of((edep)/**NP**)
do k = 1, number_of_photons_in_i_and_j
call multiple subroutines to track the photon history
(Pass in *newly create temporary tally variables to above subroutines and update tallies in them)
(Do OpenMP reduction operation on these newly created temporary tally variables returned back from above subroutines)
end do
end do
end do
.

Call to various MPI_REDUCE(..) to add up tally variables.

The OpenMP version of the code required additional modifications than simply replacing Tera directives with equivalent OpenMP directives. The tally variables are shared variables, located in COMMON blocks, and to insure proper updates of these tally variables we declared them as OpenMP reduction variables. When a subroutine (such as the subroutines called within the inner most loop over number_of_photons_in_i_and_j) manipulates a variable that exists in a COMMON block, it does not affect the private copy. Hence it was necessary to create additional temporary tally variables (for each of the tally variables that were updated in the subroutines called within the inner most loop over number_of_photons_in_i_and_j) and pass them into these subroutines. OpenMP reduction operations done on these new temporary tally variables insured determinacy.

MPI_REDUCE operations were done among MPI tasks across nodes to complete the accumulation of all the tally variables created in each node. In summary the hybrid version distributes total number of photons equally among nodes, and within a node distributes the zones among processors.

7. Results and discussion

The physical problem simulated is of photon transport through an ICF plasma consisting of a 50%-50% mixture of deuterium and tritium (D-T) at elevated temperature and density. This mixture is surrounded by a SiO_2 region, also at elevated temperature and density. A single time step is modeled, during which approximately 24,000,000 photons are emitted in both regions. The regions are divided up into 1,960 zones, arising from 49 axial mesh intervals and 40 radial mesh intervals. Twelve energy groups are used.

Tables 1,2,3, and 4 contain TPHOT timing results on various parallel machines. We provide the wall clock execution time, the speedup, and the efficiency. Speedup is the ratio of the code execution time on one processor to that on multiple processors. Efficiency is defined as the speedup divided by the number of processors.

Table 1 gives performance results for solving the test problem with TPHOT on the MTA using the two different strategies as explained in section 4. Tables 2 and 3 give performance results of TPHOT, parallelized using MPI, on the T3E and the SP respectively. On a single-processor, the MTA is about four times faster than the T3E and about 2.5 times faster than the SP. This is partly due to poor cache reusability of the code which, being a Monte Carlo code, has many conditional branch statements. Also on the MTA even for one processor parallel execution takes place due to multithreading. Each MTA processor typically used 60 threads. On the MTA, the second strategy of parallelizing across fused loops of zones and energies shows better performance even on single processor. The reason for this is that switching loop indices from higher to lower allowed better load balance among 60 threads, within one processor, since the volumes of the zones became larger at the beginning and smaller at the end. Number of photons emitted in a zone is proportional to the size of the zone.

Scalability on the T3E is nearly linear to 64 processors. Scalability on the SP is also almost linear to 64 processors. We notice minimal effect of the slow switch on the SP machine since the MPI implementation of TPHOT has very little communication. For the MPI implementation of TPHOT, used on the T3E and the SP, parallelization is done across the 24,000,000 photons. Even on 64 processors, the work per processor (=24,000,000/64 =375,000 photon simulations) is substantial. Since the computation is embarrassingly parallel and there is little communication between processors, scalability is linear. The good scalability on the T3E and the SP is possible because the geometry and material properties for all the zones fit in the local memory of each processor. For much larger problems this may not be possible. Scalability on the MTA is poor when parallelization is only by zones, but nearly linear to 8 processors when parallelization is by both zones and energies. Poor scalability on the MTA when parallelized across the 1,960 zones is due to insufficient parallelism and poor load balance. For the 8-processor case, each processor covers 245 (=1,960/8) zones and simulates all 24,000,000 photons in these 245 zones. On the MTA typically 60 threads are used per processor. This allows each thread to cover on the average about 4 (= ~245/60) iterations. This does not provide enough work for all the threads in a processor to hide latency efficiently. Parallelizing over fused

0-7695-0574-0/2000 $10.00 © 2000 IEEE

zones and energy groups resulted in total iteration of 23520 (=1960*12) for the outer loop. This allowed each processor to cover 2940 (=23520/8) iterations. Clearly this modification provided enough iteration for all the threads, even for 8 MTA processors, to hide latency and amortize overheads, and hence resulted in near perfect scalability.

Next, we discuss the parallel performance of the hybrid MPI-OpenMP implementation of TPHOT on the SMP IBM SP. These results are given in table 4. We notice that the OpenMP parallel performance across processors within a node is near perfect for TPHOT. This is evident from the timing and speedup data shown in the first four rows, with numerical data, of table 4 where number of threads increase from 1 to 8. We implemented both of the shared memory parallelization approaches, as explained in section 4.1 and 4.2, using OpenMP. Unlike the MTA there was no difference in performance between these two approaches on the shared memory processors, within a node, of the SP. This is expected on the SP since even for the parallelization over zones there is enough iteration (1960) for 8 OpenMP threads of the SP processors within a node. The scaling of the MPI-OpenMP implementation across nodes is almost linear as evident from the timing results shown in the last five rows of table 4.

The hybrid MPI-OpenMP version of TPHOT is the most complex, in terms of programming, among the three parallel versions. OpenMP is also prone to false sharing when an array of data is accessed and updated by all the processors of a node and may become the bottleneck. Currently OpenMP compilers are also less robust than MPI compilers and hence require careful programming and testing. However, this hybrid programming model allows to use the whole shared memory of a SMP node. It is necessary to use the hybrid MPI-OpenMP mode of TPHOT on the SMP SP since the large share memory available within a node will be required to simulate very large problems. Comparing the scaling results between tables 3 and 4 it appears that for large simulations, which use very large number of nodes, the hybrid model might outperform the straight MPI model on SMP IBM SP.

Acknowledgments

John Feo of Tera Computer company provided many valuable suggestions. Author also acknowledges the computational resources at the San Diego Supercomputer Center and the DARPA contract DABT63-97-C-0028, the NSF grant ASC-9613855, and the NSF NPACI Cooperative Agreement number ACI-9619020.

References

[1] W. R. Martin, A. Majumdar, J. A. Rathkopf, and M. Litvin, "Experiences with Different Parallel Programming Paradigms for Monte Carlo Particle Transport Leads to a Portable Toolkit for Parallel Monte Carlo," Proceedings of International Joint Conference on Mathematical Methods and Supercomputing in Nuclear Applications, Karlsruhe, Germany, Vol. II, pp. 418 (April 1993).

[2] F. W. Bobrowicz, J. E. Lynch, K. J. Fisher, and J. E. Tabor, "Vectorized Monte Carlo Photon Transport," *Parallel Computing*, **1**, pp. 295-305 (1984).

[3] W. R. Martin, P. F. Nowak, and J. A. Rathkopf, "Monte Carlo Photon Transport on a Vector Supercomputer," *IBM Journal of Research and Development*, **30**, pp. 193 (1986).

[4] W. R. Martin, T. C. Wan, T. S. Abdel-Rahman, and T. N. Mudge, "Monte Carlo Photon Transport on Shared Memory and Distributed Memory Parallel Processors," *International Journal of Supercomputer Applications*, **1** (3), pp. 57 (1987).

[5] P. J. Burns, M. Christon, R. Schweitzer, O. M. Lubeck, H. J. Wasserman, M. L. Simmons, D. V. Pryor, "Vectorization of Monte Carlo Particle Transport: An Architectural Study Using the LANL Benchmark "GAMETAB"," Proceedings Supercomputing89, Reno, Nevada (Nov. 13-7, 1989)

[6] W. R. Martin and F. B. Brown, "Status of Vectorized, Monte Carlo for Particle Transport Analysis," *International Journal of Supercomputer Applications*, **1** (2), pp. 11 (1987).

[7] See http://science.nas.nasa.gov/Software/NPB

[8] A. Snavely, L. Carter, J. Boisseau, A. Majumdar, K. S. Gatlin, N. Mitchel, J. Feo, and B. Koblenz, "Multi-processor Performance on the Tera MTA," Proceedings Supercomputing98, Orlando, Florida (Nov. 7-13, 1998)

[9] J. Boisseau, L. Carter, K. S. Gatlin, A. Majumdar, and A. Snavely, "NAS Benchamarks on the TERA MTA", Proceedings Workshop on Multi-Threaded Execution, Architecture, and Compilers (M-TEAC), Las Vegas, Nevada (January 1998).

0-7695-0574-0/2000 $10.00 © 2000 IEEE

Table 1. Parallel performance on the MTA using multithreading

Procs	Time (sec)	Speed-up	Efficiency
Parallelization by zones only			
1	764	1.00	1.00
2	400	1.91	0.95
4	227	3.37	0.84
8	167	4.58	0.57
Parallelization by zones and energies			
1	745	1.00	1.00
2	370	2.01	1.01
4	187	3.98	0.99
8	94	7.92	0.99

Table 2. Parallel performance on the T3E using MPI

Procs	Time (sec)	Speed-up	Efficiency
1	2,997	1.00	1.00
2	1,507	1.99	0.99
4	746	4.01	1.02
8	377	7.95	0.99
16	189	15.86	0.99
32	95	31.55	0.98
64	47	63.76	0.99

Table 3. Parallel performance on the SP using MPI (* used the slow switch)

# of nodes	# MPI task per node	Total # of MPI tasks	Time(sec)	Speedup	Efficiency
1	1	1	1924	1.00	1.00
1	2	2	963	1.99	0.99
1	4	4	482	3.99	0.99
1	8*	8	243	7.92	0.99
2	2	4	482	3.99	0.99
2	4	8	241	7.98	0.99
2	8*	16	123	15.64	0.97
4	2	8	242	7.95	0.99
4	4	16	121	15.90	0.99
4	8*	32	61	31.54	0.98
8	2	16	121	15.90	0.99
8	4	32	62	31.03	0.96
8	8*	64	32	60.12	0.94
16	8*	128	17	113.17	0.88

Table 4. Parallel performance on the SP using MPI-OpenMP

#of nodes	# of MPI task per node	# of OpenMP threads per node	(#of MPI task) *(#of OpenMP threads.)	Time (sec)	Speedup	Efficiency
1	1	1	1	1943	1.00	1.00
1	1	2	2	960	2.02	1.01
1	1	4	4	485	4.00	1.00
1	1	8	8	244	7.96	0.99
2	1	8	16	122	15.92	0.99
4	1	8	32	61	31.85	0.99
8	1	8	64	31	62.67	0.98
16	1	8	128	16	121.43	0.95

0-7695-0574-0/2000 $10.00 © 2000 IEEE