Journal of Computational Physics 228 (2009) 3882–3890

Contents lists available at ScienceDirect

Journal of Computational Physics

journal homepage: www.elsevier.com/locate/jcp

An efficient, robust, domain-decomposition algorithm for particle Monte Carlo

Thomas A. Brunner [a,*], Patrick S. Brantley [b]

[a] Sandia National Laboratories, P.O. Box 5800, Albuquerque, NM 87185-1186, United States
[b] Lawrence Livermore National Laboratory, P.O. Box 808, Livermore, CA 94551, United States

ARTICLE INFO

Article history:
Received 18 September 2008
Received in revised form 10 February 2009
Accepted 12 February 2009
Available online 26 February 2009

Keywords:
Monte Carlo methods
Parallel computation
Radiative transfer
Neutron transport

ABSTRACT

A previously described algorithm [T.A. Brunner, T.J. Urbatsch, T.M. Evans, N.A. Gentile, Comparison of four parallel algorithms for domain decomposed implicit Monte Carlo, Journal of Computational Physics 212 (2) (2006) 527–539] for doing domain decomposed particle Monte Carlo calculations in the context of thermal radiation transport has been improved. It has been extended to support cases where the number of particles in a time step are unknown at the beginning of the time step. This situation arises when various physical processes, such as neutron transport, can generate additional particles during the time step, or when particle splitting is used for variance reduction. Additionally, several race conditions that existed in the previous algorithm and could cause code hangs have been fixed. This new algorithm is believed to be robust against all race conditions. The parallel scalability of the new algorithm remains excellent.

1. Introduction

In any domain decomposed particle Monte Carlo code, two types of information need to be exchanged between the processors. Most obviously, the particles that cross domain boundaries need to be sent to the new processor. Additionally, all processors must coordinate exiting the particle processing loop at the end of the time step when all the particles have finished.

We have found that two key aspects make this new algorithm scalable and robust. The first is that all communications must be asynchronous. The second is that no single processor can have a disproportionate amount of communication work relative to the others. We will first describe in the abstract case how we send and receive all the messages. We will then describe in more detail the specifics of what is communicated.

Using asynchronous communication for all point-to-point communications allows maximal overlap between computational work and communication, and results in increased parallel efficiency. Nonblocking receives are posted for all possible incoming messages. When an incoming message is detected, the data is appropriately handled and another nonblocking receive is immediately posted. All outgoing messages are also sent with the nonblocking sends – this is critical to avoid race conditions. The outgoing message is copied to a new buffer so that the processing algorithm can reuse the working buffer immediately. The outgoing buffers are freed once it has been confirmed that the message has been received.

* Corresponding author. Tel.: +1 505 844 1253; fax: +1 505 845 7820.
E-mail addresses: tabrunn@sandia.gov (T.A. Brunner), brantley1@llnl.gov (P.S. Brantley).

0021-9991/$ - see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcp.2009.02.013

Early versions of the algorithm [1,2] had all control messages sent directly to and from the root processor. In this case with more than a few tens of processors, the root processor has significantly more work managing the communications and quickly falls behind on processing particles, thus destroying scalability. This point is not new [1], but we want to reemphasize this point here as it is critical to the scalability of the entire algorithm.

All outstanding messages are checked for completion after a certain number of local particles, N_{period}, have completed, or continually if there are no particles left to simulate on the local processor. At the end of the time step, all nonblocking receive requests are canceled.

2. Description of the algorithm

The complete algorithm is shown in Algorithm 1, with details of distinct tasks within the algorithm shown in Algorithms 2–4. We will use the notation "(Line 2.7)" to refer to line 7 of Algorithm 2 when describing the details.

A key difference between this algorithm and the previous one [1] is that the number of particles created is collected in addition to the number of particles completed. This supports arbitrary physics where the number of particles to simulate is not known at the beginning of a time step. This collection of particle counts is also done in a nonblocking (or asynchronous) way, allowing maximal overlap of work and communication. Again, all communications (particle transfers, particle count collection, and control messages) are done with nonblocking, or asynchronous, messages.

Algorithm 1: The complete algorithm for a time step.

```
1  get list of neighbor processors
2  foreach neighbor do
3  |  post nonblocking receive for maximum particle buffer size (MPI_Irecv)
4  |  allocate particle buffer
5  numProcessedParticles = 0
6  localOnlyCompleted = 0
7  localPlusChildrenCompleted = 0
8  localOnlyCreated = census size
9  localPlusChildrenCreated = census size
10 calculate parent and children processor ID numbers in binary tree
11 foreach child do
12 |  post nonblocking receive for localPlusChildren particle tallies
   |  (MPI_Irecv)
13 post nonblocking receive for control message from parent (MPI_Irecv)
14 while global finished flag is not set do
15 |  if any local particles then
16 |  |  Advance Particle
17 |  |  increment numProcessedParticles
18 |  if numProcessedParticles = N_period or no active local particles then
19 |  |  Process Messages
20 |  |  numProcessedParticles = 0
21 |  if no active local particles then
22 |  |  Control Termination
23 cancel all outstanding nonblocking receives
24 free all buffers
```

2.1. Particle transfers

When a particle hits a domain boundary, it must be sent to the processor that owns the new part of the domain. The details are shown in Algorithm 2.

The particles are first added to a buffer (Line 2.7) that has a maximum size of N_{buffer}. One buffer is associated with each neighbor processor. When the maximum number of particles has been added to the buffer, it is sent to the neighboring processor (Line 2.9). The buffer is also flushed when all local particles have been processed and there is no other work to do, even if it is only partially full (Line 4.2). The actual number of particles sent is encoded into the message along with the particle data so that the receiving processor has this information (Line 3.5).

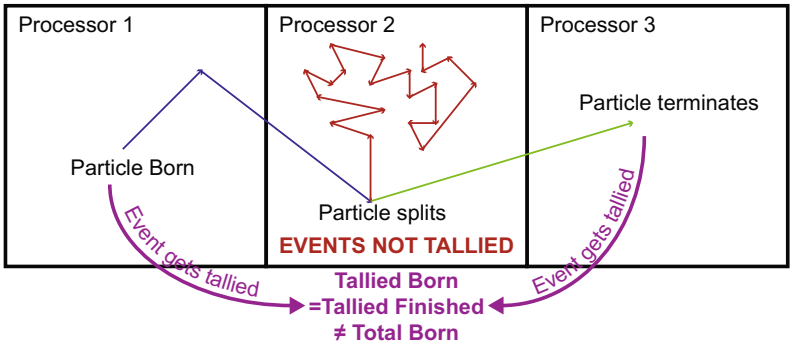

Fig. 1. One mechanism of how the nonblocking estimated global sums could lead to false completion of the time step.

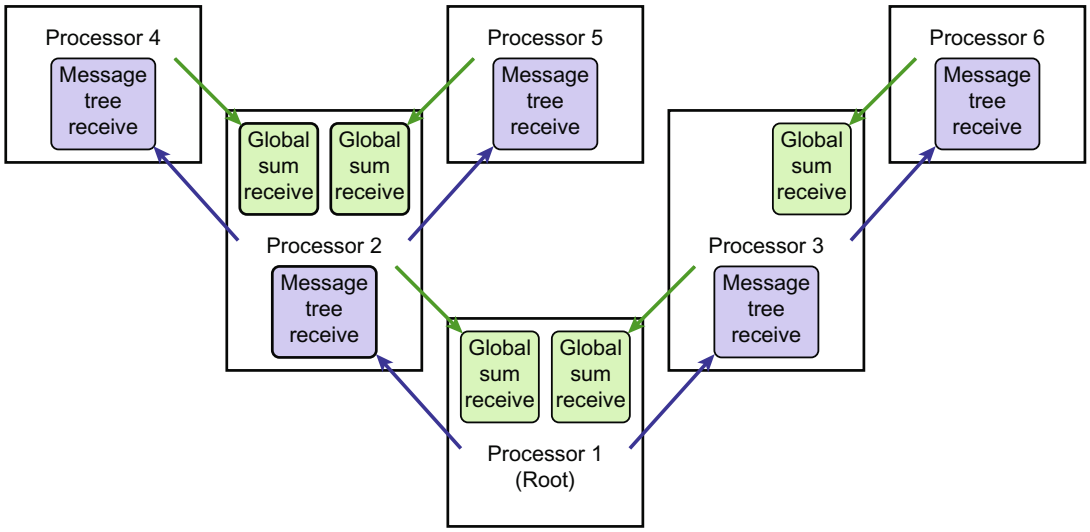

Fig. 2. The estimated global sums of particle tallies are collected down the binary tree, following the green arrows. Control messages for initiating a blocking sum or terminating the time step flow up the tree, following the blue arrows.

Note that two sets of tallies for the number of created and completed particles are needed on each processor. The first set, *localPlusChildrenCreated* and *localPlusChildrenCompleted*, is for the estimated nonblocking sum; their counts are reset to zero (Line 4.17) when their values are sent to the parent and may contain the counts of all children processors. The second set, *localOnlyCreated* and *localOnlyCompleted*, is purely local, and they are used in the blocking global sum (Line 4.8). They are not reset until the next time step.

3. Results

The performance of the algorithm is studied using two test problems, the first an Implicit Monte Carlo (IMC) [3] thermal radiation transport problem and the second a particle Monte Carlo neutronics [4] test problem. Both are designed to be perfectly load balanced and keep a constant amount of work per processor in the weak scaling studies.

3.1. Problem descriptions

The IMC problem models a uniform box of material [1] in thermal equilibrium with the radiation. Each processor simulates a one centimeter cube with 30 zones per side, for a total of 27,000 zones per processor. There are on average 10 particles per zone. All external boundaries are reflecting. The box is filled with a uniform, hot material at $T = 1$ keV, a density of

$\rho = 1000\ \mathrm{kg/m^3}$, and a heat capacity of $C_v = 5 \times 10^9\ \mathrm{J/K\ kg}$. The absorption and scattering cross sections are $\sigma_a = 50\ \mathrm{cm^{-1}}$ and $\sigma_s = 10\ \mathrm{cm^{-1}}$. Ten constant time steps were computed with $\Delta t = 3 \times 10^{-9}\ s$. Besides a straight forward IMC calculation, two other modifications were also run in order to stress the algorithm for an unknown number of particles. We artificially split some of the photons during each time step. When a particle is split, the new particle's random number sequence is initialized by using a cryptographic hashing algorithm; no communication is needed to initialize the new particle. Three split fractions were used for the tests, 0%, 10%, and 50%. These problems were run on Red Storm [5], a 12,960 node Cray XT3 with one dual-core processor each, at Sandia National Laboratories.

We tested the ParticleMC Monte Carlo particle transport package [6], developed for inertial confinement fusion (ICF) applications [7], with another uniform infinite medium problem. The medium is equimolar deuterium and tritium with a temperature of $T = 25$ keV and a density of $\rho = 250\ \mathrm{g/cm^3}$, specifications representative of an ICF capsule simulation [7]. All external boundaries are reflecting. In all cases, twenty constant sized time steps were computed with $\Delta t = 10^{-11}$ s. A spatially uniform, isotropic source of neutrons with initial energy 14 MeV and a constant source rate of $5 \times 10^9\ \mathrm{s^{-1}}$, or one physical neutron total sourced into the problem, is used to represent neutrons from deuterium–tritium thermonuclear reactions. For these specifications, the zone size in the simulations described below is on the order of a mean free path at the initial source energy (analogous to the IMC simulations). The nuclear cross section data is represented using 175 neutron energy groups, although the Monte Carlo neutrons have a continuous energy value. Both deuterium and tritium can undergo a neutron-induced $(n, 2n)$ reaction resulting in the production of additional neutrons. For this test problem, approximately 2.5% of all deuterium nuclear reactions are $(n, 2n)$ reactions, while for tritium the fraction is approximately 1%. When a new neutron is produced, the random number sequence for the new particle is initialized using a cryptographic hash algorithm with no parallel communication required. Unlike the IMC problem, the number of zones and particles per zone were varied in the problems below. All ParticleMC test problems were run on Purple [8], a 1534 node IBM machine with eight Power5 1.9 GHz processors per node, at Lawrence Livermore National Laboratory.

3.2. Finding good parameter values

The algorithm has two free parameters, the maximum buffer size, N_{buffer}, and the message check period, N_{period}. Each test problem was run on 64 processors and each parameter was varied from 1 to 16,384 to find the minimum run time for these problems. For the IMC test, we used the physical parameters listed in the previous section. Each run for the IMC was only done once. For the particle Monte Carlo, the infinite medium was simulated using a unit cube with reflecting boundaries. The mesh used 50 zones per side (125,00 total zones) and was decomposed into 64 domains. A total of 10^7 Monte Carlo neutrons were followed per time step (80 particles per zone, on average). Each parameter combination was run five times and the results averaged. The results are shown in Fig. 3.

Essentially the same result was obtained for significantly different transport physics on two different machines. When either N_{buffer} or N_{period} were set to one, too much time is spent doing communication work, and the total run time is very long. Away from these obviously poor choices, the run time is a very shallow function of either parameter. The parameters that gave the minimum run time were N_{period} = 16,384 and $N_{\mathrm{buffer}} = 512$ for the IMC and $N_{\mathrm{period}} = 1024$ and $N_{\mathrm{buffer}} = 1024$ for particle Monte Carlo. We used these values in all the subsequent runs below.

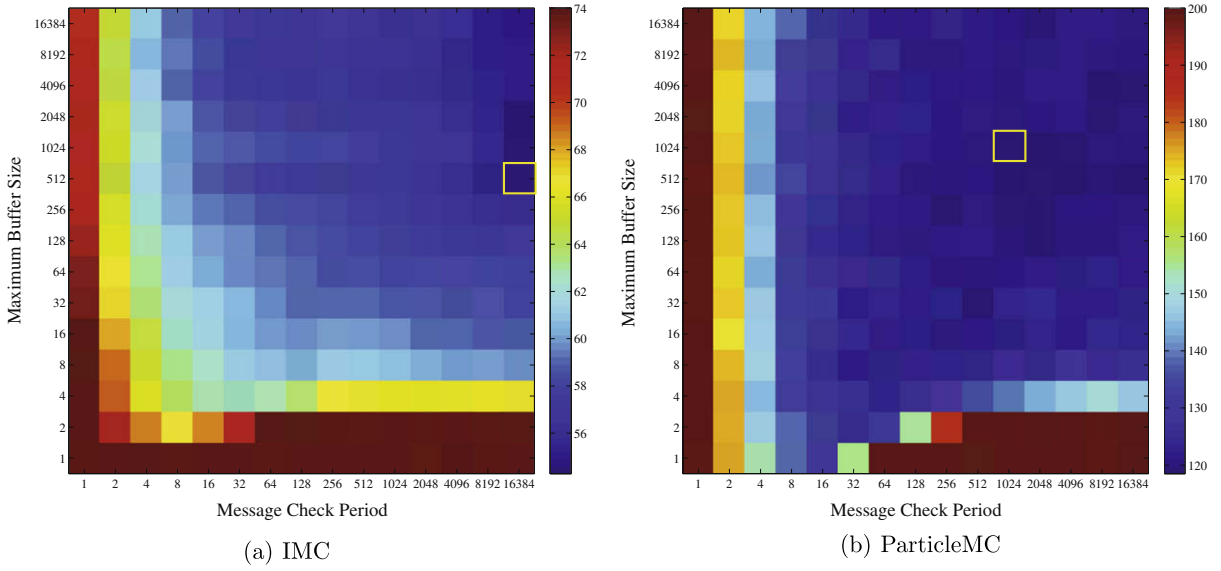

Fig. 3. Run time as a function of both message check period and maximum buffer size. The minimum is indicated by the yellow box, at N_{period} = 16,384 and $N_{\mathrm{buffer}} = 512$ for IMC and $N_{\mathrm{period}} = 1024$ and $N_{\mathrm{buffer}} = 1024$ for ParticleMC.

The run time generally improved as the completed messages were checked less frequently. This is because it is generally better to do real work while there is work to do, and save the communication work until the end. On the other hand, relatively short buffer sizes seemed to be better than very long messages. With smaller buffer sizes, work is made available to neighbors sooner, which can help in load-imbalanced cases. The old algorithm [1] would lock if the message check period was larger than the maximum buffer size.

While the absolute ideal set of parameters is likely machine- and problem-dependent, in practice we have noticed that any set of reasonably-chosen values performs well for a wide range of problems. In fact, both sets of runs gave very similar results, despite simulating different physics and running on different computer architectures.

3.3. A strong scaling study

Next we tested each package in a strong scaling study, where a constant sized problem is spread onto more and more processors. The amount of real work stays constant and the amount of communication work increases. For this study, the particle Monte Carlo package used the same mesh and number of Monte Carlo particles as in the parameter scan above. Fig. 4 shows the results of increasing the number of spatial domains.

Initially, the efficiency for both physics packages rises above 100% until approximately 125–216 processors. In order to understand how the efficiency increased, we ran a series of serial-only calculations, where the domain size was shrunk. Each simulation size corresponded to one of the sizes in the IMC domain decomposed strong scaling study. Ideally for this set of serial calculations, the run time should decrease linearly with the number of zones and particles in the domain, but the run time actually decreased faster. We attributed the super-linear speedup to more of the problem fitting into the faster cache memory of the processor, which reduces the need for the processor to access the slower, main memory. We then plotted this set of serial runs in Fig. 4 as if they were part of the strong scaling study, since they represent the ideal speedup of the problem if there was no communication present.

Eventually, however, the communication dominates the simulation time, and the efficiency drops below 100% for the 343 processor runs. At 1000 processors, the IMC simulation only has 27 zones and 270 photons per processor, while the particle Monte Carlo simulation has 125 zones and 10,000 Monte Carlo neutrons per processor. Since the initial particle Monte Carlo calculation was bigger than the IMC calculation, it has more work to do per processor for a given decomposition, so it scales slightly better. Increasing the arbitrary split fraction for the IMC problem also increases the amount of work per processor and thus increases its efficiency. Despite simulating different transport physics on different machines, the algorithm performs very similarly for both problems.

3.4. A weak scaling study

We also tested the algorithms in a weak scaling study, where the amount of real work per processor remains constant. In this case, we do this by increasing the problem domain. Each processor simulates a one centimeter cube of the infinite medium. The particle Monte Carlo package had 25 zones in each direction of the mesh with 640,000 Monte Carlo neutrons per domain per time step. The IMC was tested with the same cube as described above. The results are shown in Fig. 5.

In both physics, the efficiency drops off until 27 processors are employed. Because the domains in this case are arrayed spatially in a three by three cube, this is the first configuration in which one processor communicates with six

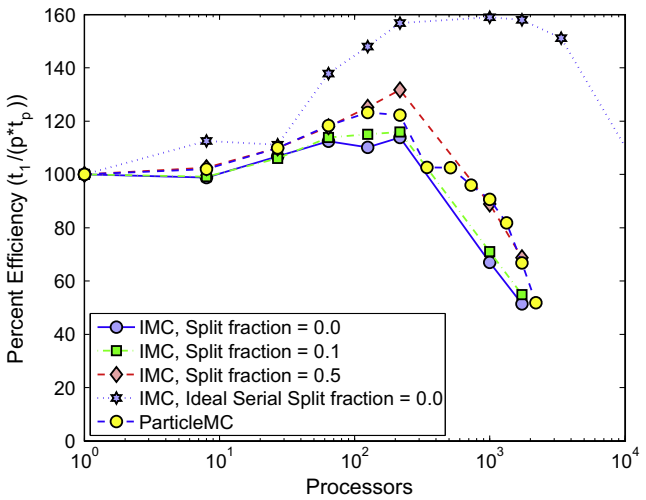

Fig. 4. Parallel efficiency for a strong scaling study where a fixed amount of work was spread on different numbers of processors.

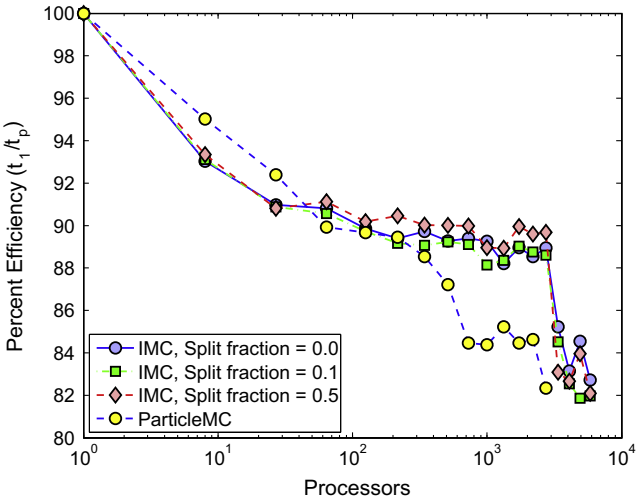

Fig. 5. Parallel efficiency for a weak scaling study where the work per processor was kept nearly constant for IMC, using three different particle split fractions, and ParticleMC. Note that the scale only goes down to 80% to amplify the differences in the different simulations.

neighbors. After this point the work to communication remains constant. And indeed, the scaling stays roughly constant for the IMC at about 88% until we get to many thousands of processors. The split fraction in the IMC simulations had virtually no effect on the scalability of the algorithm. At 3375 processors, there is an unexplained drop in the efficiency, but the efficiency stays above 80% to nearly 6000 processors. In all cases there was only one blocking global sum executed; in none of our tests could we force the estimated non-blocking sums to trigger a blocking sum before the time step was actually complete. (We have seen this happen in much more complicated user problems, however.) The particle Monte Carlo efficiency drops a little sooner. In addition to being run on a different machine, the processors were not completely load balanced, contrary to the problem design. The algorithm used to decompose the domain was designed to work for purely unstructured problems, and it is difficult to make it decompose the problem into pure cubes. For example, in the 729 processor case, where the particle Monte Carlo has the drop-off, the zone count per processor varies between 11,812 and 16,642. Nonetheless, the weak scaling for the ParticleMC problem remains above 80% up to 2744 processors.

4. Conclusions

This new algorithm is much more robust than the one described previously [1,2]. It scales well for load balanced problems, but this algorithm will not scale well on problems that are not load balanced. Other techniques, such as a combination of domain replication and domain decomposition or dynamic mesh load balancing, will need to be employed to boost the parallel efficiency on real problems. These techniques are currently under investigation.

Additionally, when this algorithm was employed in real problems using IMC physics, we noticed significant run-time variability with the parameters of $N_{\text{buffer}} = 512$ and $N_{\text{period}} = 16{,}384$. If we changed the values to be similar to the values determined for the particle Monte Carlo, the run-time variability was significantly reduced. We currently speculate that MPI or the underlying network could not support the number of outstanding messages that the original parameters yielded, and checking more frequently and sending fewer buffers (because they are bigger) reduced the number of messages. This needs to be investigated further, however.

Acknowledgment

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under Contract DE-AC04-94AL85000.

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

References

[1] T.A. Brunner, T.J. Urbatsch, T.M. Evans, N.A. Gentile, Comparison of four parallel algorithms for domain decomposed implicit Monte Carlo, Journal of Computational Physics 212 (2) (2006) 527–539.

[2] N.A. Gentile, M. Kalos, T.A. Brunner, Obtaining identical results on varying numbers of processors in domain decomposed particle Monte Carlo simulations, in: F. Graziani (Ed.), Computational Methods in Transport: Granlibakken 2004, Lectures Notes in Computational Science and Engineering, vol. 48, Springer, 2006. also UCRL-PROC-210823.
[3] J.A. Fleck Jr., J.D. Cummings, An implicit monte carlo scheme for calculating time and frequency dependent nonlinear radiation transport, Journal of Computational Physics 8 (1971) 313–342.
[4] E.E. Lewis, W.F. Miller Jr., Computational Methods of Neutron Transport, American Nuclear Society, Lagrange Park, Illinois, 1993.
[5] Introducing Red Storm. <http://www.sandia.gov/ASC/redstorm.html>.
[6] P.S. Brantley, L.M. Stuart, Monte carlo particle transport capability for inertial confinement fusion applications, in: Joint International Topical Meeting on Mathematics & Computation and Supercomputing in Nuclear Application (M&C+SNA 2007), American Nuclear Society, Monterey, California, 2007.
[7] J.D. Lindl, Inertial Confinement Fusion, Springer-Verlag, New York, 1998.
[8] Advanced simulation and computing: Purple. <http://asc.llnl.gov/computing_resources/purple>.