SAND2007-6766C

An Efficient, Robust, Domain-Decomposition Algorithm for Particle Monte Carlo, invited

Thomas A. Brunner

Sandia National Laboratories

INTRODUCTION

A previously described algorithm[1, 2] for doing domain decomposed particle Monte Carlo calculations in the context of thermal radiation transport[3] has been improved. The old algorithm did not support cases where the number of particles at the beginning of the time step is unknown, for example when particle splitting is used for variance reduction. Additionally, several race conditions existed in the old algorithm that caused periodic code hangs. This new algorithm is believed to be robust against all race conditions. Even with these two changes, scalability remains excellent.

OUTLINE OF THE ALGORITHM

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.

Asynchronous communication is used for all point-to-point communications. This allows maximal overlap between computational work and interprocessor communication, and increases 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 buffers are freed once it has been confirmed the message has been received. All outstanding messages are checked for completion after a certain number, N_{period}, of local particles 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.

Particle Transfers

When a particle hits a domain boundary, it must be sent to the processor that owns the new part of the domain. These particles are sent to a buffer that has a maximum size of N_{buffer}. There is one buffer for each neighbor processor. When the maximum number of particles has been added to the buffer, it is sent to the neighboring processor. The buffer is also flushed when all local particles have been processed and there is no other work to do, even if it is partially full. The actual number of particles sent is encoded into the message along with the particle data so that the receiving processor knows how many particles were actually sent.

Each processor also posts a nonblocking receive for each neighbor processor, with enough storage for the maximum number of particles. When an incoming message is detected, the buffer is processed and the particles put on the simulation particle list.

Coordinating the end of the time step

In order to determine that all that particles have finished, the number of particles created and completed during a time step on each processor are tallied. When the instantaneous global sum of these two tallies match, the time step is over. This is only possible to do with a blocking global sum, but these can be expensive since they require synchronization between all processors.

Instead we first estimate the global sum with a nonblocking sum that asynchronously collects the tallies from all processors to the root processor, which also does particle computations. This asynchronous global sum will eventually be exact, but situations may arise where the global sums as seen by the root match, but there are unprocessed messages that would make the sums not match. A binary tree communication pattern is set up where each processor, except the root, has one parent and at most two children. Each processor posts nonblocking receives from its children to collect the particle counts. When an incoming message is received, the off-processor counts are added to the local processor's counts. When there is no more local work to do, each processor sends the counts to its parent in the binary tree, and the local counts are reset.

When the estimated global sums of the two tallies match on the root processor, a message is sent up the binary tree from parents to children signaling each processor that the time step might be over and to initiate a blocking global sum, even if a local processor is still working on particles. If the counts in the blocking global sums match, the time step is over. The estimate nonblocking global sums continue until its sums match the blocking sums — all messages from the nonblocking send need to be collected or else they will be erroneously received in the next time step. When all the counts match, another message is sent through the nonblocking communication tree telling each processor that the time step is really finished now, and all processors exit the processing loop. Until this message is received, processors keep checking for "maybe done" message, incoming particles from neighboring processors, or processing the asynchronous sum messages. If the blocking global sum counts don't match, the processing continues as before, waiting until the next time the estimated nonblocking global sums match.

Note two sets of tallies for the number of created and completed particles are needed on each processor. The first is for the estimated nonblocking sum, and the counts are reset to zero when their values are sent to the parent and may contain the counts of all children processors. The second set is for the blocking global sum and is not reset until the next time step and is a purely local count.

RESULTS

The performance of the algorithm is studied using an equilibrium, uniform box of material[1]. This is designed to be perfectly load balanced, and to nearly keep a constant amount of work per processor in a weak scaling study. 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 $T = 1\,\text{keV}$, a density of $\rho = 1000\,\text{kg/m}^3$, and a heat capacity of $C_v = 5 \times 10^9\,\text{J/K kg}$. The absorption and scattering cross sections are $\sigma_a = 50\,\text{cm}^{-1}$ and $\sigma_s = 10\,\text{cm}^{-1}$. Ten time constant sized steps were computed with $\Delta t = 3 \times 10^{-9}$s. All test problems were run on Red Storm[4], a 12,960 node Cray XT3, at Sandia National Laboratories.

The problem was first run on 64 processors with different maximum sized buffers, N_{buffer}, and different periods, N_{period}, to minimize the run time for this uniform problem. Each parameter ranged from 1 to 16384. The results in Figure 1 indicate that a longer check period is better, and a maximum buffer size of about 512 particles resulted in the lowest run time for this problem. Run time appears to be a very shallow function of both parameters; we have seen in practice that any set of reasonably chosen values to preform well for a wide range of problems. The long check period indicates as long as there is local work, a processor should spend as little time as possible checking for incoming messages. The old algorithm[1] would lock if the message check period was larger than the maximum buffer size.

Using the parameters that resulted in the minimum run time in the last test, a scaling study was performed for three different cases. In order to simulate the effects of variance reduction techniques on the algorithm, the photons were arbitrarily split with three different probabilities of 0.0, 0.1, and 0.5. Figure 2 shows the efficiency is nearly constant at about 85% after about 27 processors. The efficiency drops off until 27 processors where one processor has six neighbors for the first time, and after this point the work to communication remains remains constant. The algorithms is slightly less efficient for the higher split fractions because there are about 50% more messages sent between processors, but this this reduction

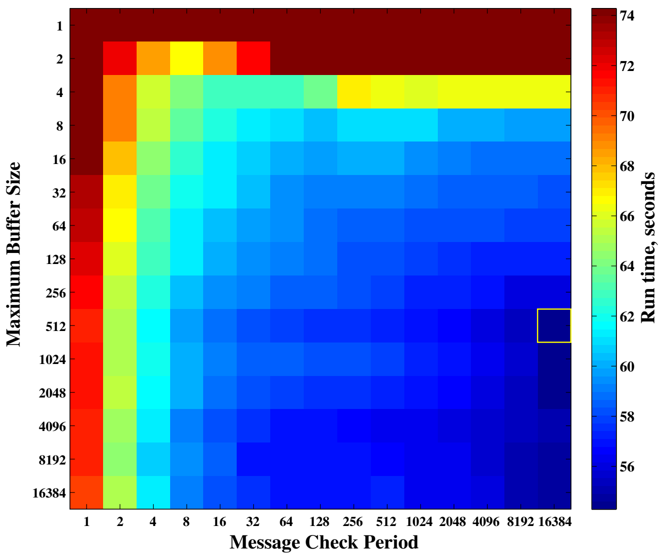

Fig. 1. Run time as a function of both message check period and maximum buffer size. The minimum is indicated by the yellow box at $N_{\text{period}} = 16386$ and $N_{\text{buffer}} = 512$.

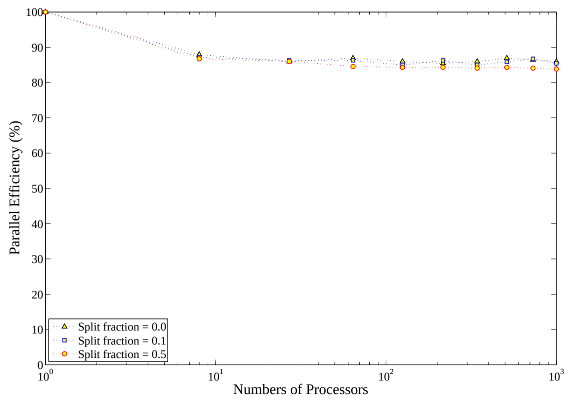

Fig. 2. Parallel efficiency for a scaling study where the work per processor was kept nearly constant using three different particle split fractions.

in efficiency is minor.

CONCLUSIONS

This new algorithm is much more robust and is capable of handling cases where the number of particles is unknown at the beginning of the time time step. It scales well, but this is just part of the full algorithm. The test problem was specifically designed to be perfectly load balanced. 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 load balancing, will need to be employed to boost the parallel efficiency on real problems.

ACKNOWLEDGMENTS

I would like to thank Thomas Gardiner of Cray Inc. for many helpful discussions about this work as well as running all the scaling studies on Red Storm. Patrick Brantely of LLNL helped make this algorithm much more robust by implementing it his own code, asking questions, and kindly informing me that the old algorithm, and various improvements along the way to this algorithm, still locked up on his test problems.

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.

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, pp. 527 (2005).

[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, editor, *Computational Methods in Transport: Granlibakken 2004*, volume 48 of *Lectures Notes in Computational Science and Engineering*. Springer (2006).

[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**, pp. 313 (1971).

[4] "Introducing Red Storm." URL `http://www.sandia.gov/ASC/redstorm.html`.