UCRL-CONF-208746

Comparison of Four Parallel Algorithms For Domain Decomposed Implicit Monte Carlo

T.A. Brunner, T.J. Urbatsch, T.M. Evans, N.A. Gentile

December 22, 2004

American Nuclear Society
Auignon, France
September 12, 2005 through September 15, 2005

Comparison of Four Parallel Algorithms for Domain Decomposed Implicit Monte Carlo

Thomas A. Brunner
Sandia National Laboratories
Document Number: SAND-2004-6695C

Todd J. Urbatsch and Thomas M. Evans
Los Alamos National Laboratory
Document Number: LA-UR-05-0175

Nicholas A. Gentile
Lawrence Livermore National Laboratory
Document Number: UCRL-CONF-208746

Submitted to M&C 2005 Avignon, American Nuclear Society
January 2005

Comparison of Four Parallel Algorithms for Domain Decomposed Implicit Monte Carlo

Thomas A. Brunner[*]
Sandia National Laboratories
PO Box 5800, Albuquerque, NM 87185-1186

Todd J. Urbatsch and Thomas M. Evans[†]
Los Alamos National Laboratory
PO Box 1663, Los Alamos, NM 87545

Nicholas A. Gentile[‡]
Lawrence Livermore National Laboratory
7000 East Ave., Livermore, CA 94550

February 9, 2005

1 Introduction

Four different algorithms for domain decomposed Monte Carlo are outlined, and the performance of each is measured. These algorithms are implemented in the KULL IMC package[4] running inside of ALEGRA[1]. This package implements the Implicit Monte Carlo (IMC) scheme for thermal radiation transport of Fleck and Cummings[3].

2 Algorithms

In domain decomposed Monte Carlo, two sets of data must be communicated between the processors. The nearest neighbors must exchange particles that

[*]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.

[†]Los Alamos National Laboratory, an affirmative action/equal opportunity employer, is operated by the University of California for the United States Department of Energy under contract W-7405-ENG-36.

[‡]This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-ENG-48.

cross processor boundaries. A global communication operation must also be performed so that all the processors know when all the other processors are finished moving all the particles. The four algorithms for a time step in the IMC package vary in how they perform each of these two tasks.

2.1 The KULL IMC Based Algorithms

The original communication method in the KULL IMC package[4] used blocking sends and receives to exchange all information. Particles are moved within a processor and are buffered when they hit a processor boundary. Once all local particles have been moved, each processor exchanges the number of particles to exchange with its neighbor, then allocates memory to receive the particles, and finally processes the incoming particles. A global sum is performed to tally the total number of particles that are exchanged between processors. If there are none, the time step finishes, otherwise the exchanged particles are simulated and the process repeats.

Even though each processor needs to talk with only its neighbors, this algorithm can have a serial communication pattern in certain circumstances. For example, if you had a square problem domain cut into sixteen sub-domains, it takes twelve steps to communicate all the data. It should take only four steps since each processor has at most four neighbors. This serialization is due to the fact that each processor talks with each of its neighbors one after another, in a specific order. Even if another neighbor is ready to communicate, a processor will wait for the next one in its list.

An improved version of this algorithm uses nonblocking receives, as well as eliminating the separate message for the number of particles. This eliminates the serialization of the original algorithm.

2.2 Milagro Based Algorithms

The Milagro[5] and ALEGRA-Milagro algorithms both are outlined in Algorithm 1. For particle transport in the domain decomposition topology, each processor continuously loops until every particle finishes. After simulating N active particles, the communicator is checked to see if any particles have arrived from neighboring domains on other processors. If incoming particles have arrived, they are put into the active particle list in a last-in, first-out manner. During transport, particles that leave the processor's domain are buffered and eventually sent to the appropriate processors. When a processor has no more source particles or incoming particles, it deliberately flushes its buffers, sending only the number of bytes needed to transfer the partially full buffer. The number of particles that are being sent are encoded into the beginning of the message buffer, and extracted when the buffer is received. When there appears to be no more incoming particles, the processor makes any updates to the global count of finished particles. When all particles have been completed, the master processor broadcasts the finished status to all the processors.

The original Milagro algorithm set $N = 1$ and checked for incoming messages frequently. This setting was necessary on the SGI Octane "Bluemountain" supercomputer (now decommissioned) at Los Alamos National Laboratory. Even for $N = 2$, the processors would lock occasionally on that machine. The Milagro algorithm also looped over MPI requests, making multiple calls to (`MPI_Test`).

The original Milagro also uses a fat communication tree for the "particles completed" messages, where processor zero is the parent of all other nodes. This means that processor zero checks for many "particles completed" messages, which causes a load imbalance and poor scaling.

An improved version of the Milagro algorithm uses a binary tree communication pattern[2] for the asynchronous communication for the "particles completed" messages and the finished message broadcast. This pattern ensures that each processor has an even and minimal workload for incoming message tests.

3 Performance Results

The perfectly load balanced problem is a cube with one centimeter long sides and is discretized with sixty zones per side for a total of 216,000 zones in the mesh. All boundaries are reflecting. The box is filled with a uniform, hot material at $T = 1.1604505 \cdot 10^7$ K , with an absorption cross section of $\sigma_a = 5000\,\mathrm{m}^{-1}$, with a scattering cross section of $\sigma_s = 1000\,\mathrm{m}^{-1}$, with a density of $\rho = 1000$ kg/m^3, and a heat capacity of $C_v = 5 \cdot 10^9$ J/K kg. Ten time steps were computed, each with a constant size of $\Delta t = 3 \cdot 10^{-9}$ sec. With these parameters, the effective scattering cross section of the Fleck and Cummings method is approximately $\sigma_{\text{eff}} = 6000\,\mathrm{m}^{-1}$, or one mean free path per mesh zone.

Timings include only the particle transport section of the code and not input, output, or startup costs. Each simulation was run three times, and the minimum time was used to calculate the efficiencies. The simulations were run on a Linux cluster with dual 3.05 GHz Pentium Xeon nodes and Myrinet interconnects between the nodes. Both processors on a compute node were used. In all the results, there is a drop in efficiency from one to two processors mainly due to the fact that the memory bandwidth is shared between the processors. A buffer size of 5000 particles and a message check period of 100 particle was used for the ALEGRA-Milagro algorithm.

Figure 1 shows the constant work efficiency of the four algorithms with four million particles. The constant work parallel efficiency is

$$\epsilon_{\text{CW}} = \frac{t_1}{n t_n}, \tag{1}$$

where n is the number of processors, t_1 is the serial run time, and t_n is the run time of the n processor run. The mesh was decomposed into roughly cubic chunks. The original KULL and Milagro algorithms do not scale well, each for

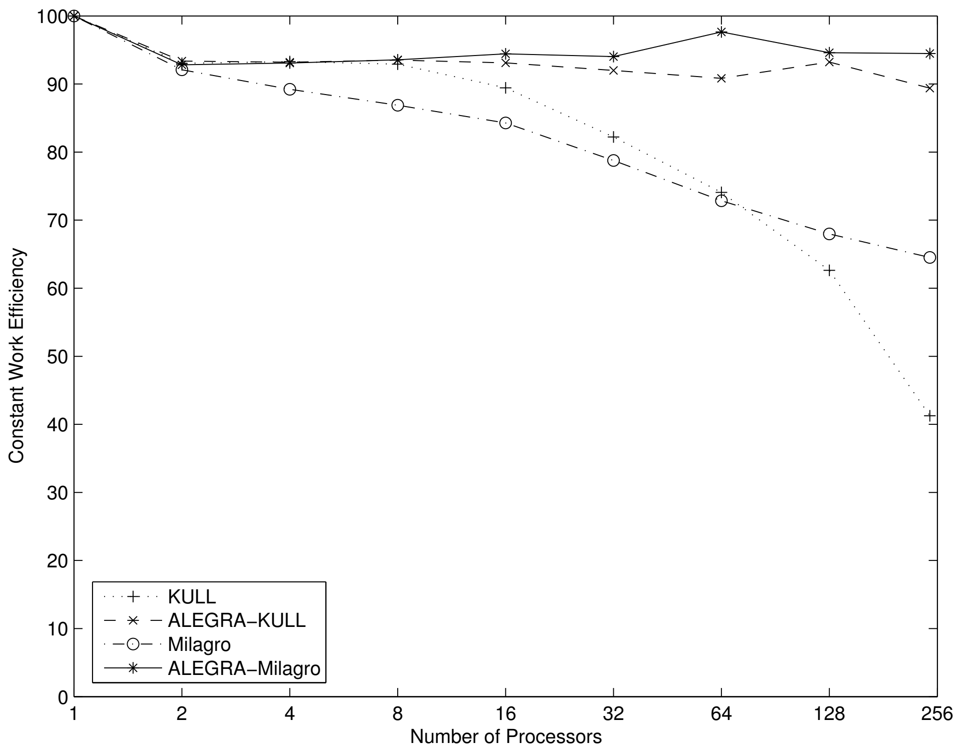

Figure 1: Parallel Efficiency, ϵ_{CW}, for the constant work hot box problem.

a different reason. The KULL algorithm has a serialized communication pattern, as discussed in Section 2.1. In the Milagro algorithm the master processor spends a lot of time checking for messages from all other processors. This leads to a significant load imbalance as the number of processors is increased. The ALEGRA-KULL algorithm scales very well, but begins to suffer from the multiple global communications within each time step. The ALEGRA-Milagro algorithm scales nearly perfectly to 244 processors. In fact, the biggest performance decrease happened between one and two processors and is more a result of machine architecture than of the algorithm behavior.

4 Conclusions

Two production algorithms for asynchronous parallel Implicit Monte Carlo radiation transport were analyzed and improved. The improved version of the Milagro algorithm, the ALEGRA-Milagro algorithm, performed the best by scaling nearly perfectly out to 244 processors on a Linux cluster. The improvements were check for messages less frequently and to use a scalable, non-blocking version of the standard reduce and broadcast functions. It is critical not to have one processor do more work than the others, even if it seems like it is a trivial amount of work, such as checking for incoming messages. The algorithms that used blocking communication do not perform well due to unnecessary contention for processor time.

References

[1] Thomas A. Brunner and Thomas A. Mehlhorn. A user's guide to radiation transport in ALEGRA-HEDP, version 4.6. Technical Report SAND2004-5799, Sandia National Laboratories, Albuquerque, NM, November 2004.

[2] Derrick Coetzee. Binary tree. Wikipedia article.

[3] J. A. Fleck, Jr. and J. D. Cummings. An implicit monte carlo scheme for calculating time and frequency dependent nonlinear radiation transport. *Journal of Computational Physics*, 8:313–342, 1971.

[4] Nicholas A. Gentile, Noel Keen, and Jim Rathkopf. The KULL IMC package. Technical Report UCRL-JC-132743, Lawrence Livermore National Laboratory, Livermore, CA, 1998.

[5] Todd J. Urbatsch and Thomas M. Evans. Milagro version 2, an implicit Monte Carlo code for thermal radiative transfer: Capabilities, development, and usage. Technical Report LA-14195-MS, Los Alamos National Laboratory, January 2005.

Algorithm 1 ALEGRA-Milagro

```
get list of neighbor processors
for each neighbor
    post nonblocking receive for maximum buffer size (MPI_Irecv)
get children processors
for each child
    post nonblocking receive for particles completed (MPI_Irecv)
get parent processor
post nonblocking receive for finished message from parent (MPI_Irecv)
Sum to master total number of particles (MPI_Reduce)
while not finished
    if any local particles
        move the last particle in the list to a termination event
        if particle hit processor boundary
            buffer particle
            if buffer full
                send particle buffer to neighbor (MPI_Send)
        else
            increment local particles completed counter
    for every $N$ particles or if no active particles
        for each incoming particle buffer (MPI_Testsome)
            unpack number of incoming particles from buffer
            process particles, adding to end of list
            repost nonblocking receive (MPI_Irecv)
        for each completed particle message from children (MPI_Testsome)
                add to local number of particles completed
                repost nonblocking receive for particles completed (MPI_Irecv)
    if no active particles
        send any partially full particle buffers (MPI_Send)
        send number of particles completed to parent(MPI_Send)
        if not master processor
            reset local particles completed to zero
        if master processor
            if all particles completed
                set finished flag
                for each child
                    send finished message (MPI_Send)
        else if finished message from parent (MPI_Test)
            set finished flag
            for each child
                send finished message (MPI_Send)
cancel all outstanding nonblocking receives
```