LA-UR- 98-2135

Approved for public release; distribution is unlimited.

Title: PARALLEL IMPLICIT MONTE CARLO IN C++

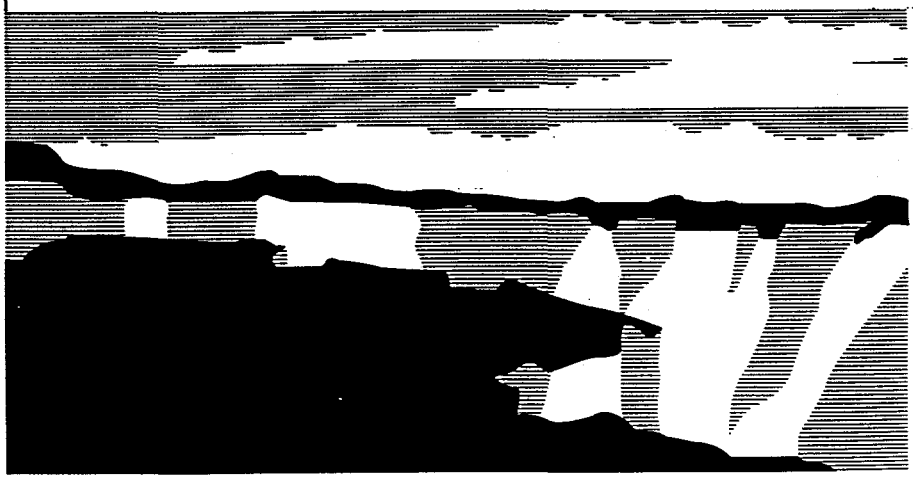
CONF-981207--

Author(s): Todd J.Urbatsch
Thomas M. Evans

Submitted to: ISCOPE'98/Int'l Symp. on Computing in Object Oriented Parallel Environments
Santa Fe, NM
December 8-11, 1998

MASTER

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED



Los Alamos
NATIONAL LABORATORY

Los Alamos National Laboratory, an affirmative action/equal opportunity employer, is operated by the University of California for the U.S. Department of Energy under contract W-7405-ENG-36. By acceptance of this article, the publisher recognizes that the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or to allow others to do so, for U.S. Government purposes. Los Alamos National Laboratory requests that the publisher identify this article as work performed under the auspices of the U.S. Department of Energy. The Los Alamos National Laboratory strongly supports academic freedom and a researcher's right to publish; as an institution, however, the Laboratory does not endorse the viewpoint of a publication or guarantee its technical correctness.

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

LA-UR- 98-2135

Parallel Implicit Monte Carlo in C++

Todd J. Urbatsch* and Thomas M. Evans**

Los Alamos National Laboratory, Los Alamos, NM 87544, USA

Abstract. We are developing a parallel C++ Implicit Monte Carlo code in the Draco framework. As a background and motivation for our parallelization strategy, we first present three basic parallelization schemes. We use three hypothetical examples, mimicking the memory constraints of the real world, to examine characteristics of the basic schemes. Next, we present a two-step scheme proposed by Lawrence Livermore National Laboratory (LLNL). The two-step parallelization scheme we develop is based upon LLNL's two-step scheme. Our two-step scheme appears to have greater potential compared to the basic schemes and LLNL's two-step scheme. Lastly, we explain the code design and describe how the functionality of C++ and the Draco framework assist our development of a parallel code.

1 Introduction

Our group at Los Alamos National Laboratory has a policy that all computer codes be born parallel. Therefore, the decision to rewrite our Implicit Monte Carlo (IMC) code in C++ required us to develop a parallelization strategy before writing any code.

Our code solves time-dependent, non-linear radiative thermal transfer problems using Monte Carlo. The Monte Carlo particles represent bundles of photons that interact with material. Material absorbs radiation, heats up, and gives off more radiation. The physical system is highly non-linear because the characteristics of the material, namely absorption and emission, are highly dependent upon temperature. Implicit Monte Carlo, or IMC, refers to a time-implicit, linearized method developed by Fleck and Cummings [1] in which, most notably, the absorption and re-emission of radiation is represented by an effective scatter. As with deterministic methods, an implicit time discretizion produces a more stable algorithm.

We present three basic parallelization schemes: full replication, full domain decomposition, and general domain decomposition/replication. Next, we present a scheme proposed by Lawrence Livermore National Laboratory (LLNL), which is a two-step scheme derived from the basic schemes. We then present our two-step scheme, which is based upon LLNL's two-step scheme, but appears to have potential for larger speedups.

* tmonster@lanl.gov
** tme@lanl.gov

Our code is written within the Draco framework [2], making use of several tools, such as smart pointers and communication classes. Both C++ and Draco have facilitated our development of a parallel IMC code.

2 Parallelization Strategies

Parallel speedup is the ratio of serial run time to multiple-processor run time for a given calculation. A calculation using 10 processors should run 10 times faster than a calculation using one processor. Theoretically, this "linear speedup" (linear with unity slope) is the best we can do. Unfortunately, overhead costs make it impossible to reach full linear speedup. Our goal – the goal of any parallelization – is to get as close to linear speedup as possible.

In our calculations, as shown in Figure 1, a host code begins the time step and then spawns the IMC calculation, which is performed on many processors. When the IMC is finished, it sends data back to the host code. We only consider the host code on one processor, but it too may be on multiple processors. The distribution of data before the IMC step and the collection of data after the IMC step constitute pre-cycle and post-cycle overhead. These communications are inherently serial and detract from the potential speedup.

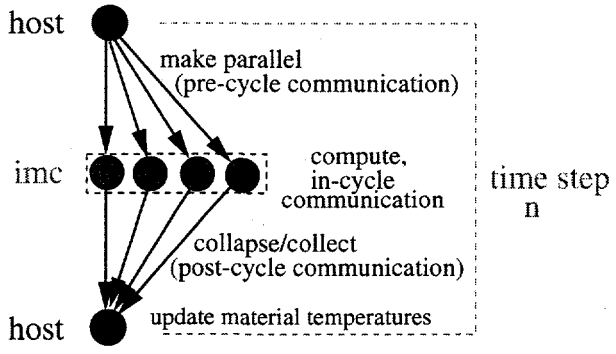


Fig. 1. Overview of a time step in a parallel IMC calculation.

To parallelize IMC, we must determine which discretized variables we want to parallelize. Whereas deterministic calculations can be parallelized in space (cells), angle (discrete ordinates), energy or frequency (groups), or some combination of those independent variables, Monte Carlo calculations may be parallelized in space (cells), particles, possibly energy or frequency (groups), or some combination of these. The best choice of variable to parallelize is one whose individual elements are independent of each other so processors need not talk to each other during the IMC step. Combine processor independence with good load balancing (where processors have the same amount of work to do), and speedups in the IMC step, apart from pre- and post-cycle communication, will be good. We

will see that the two-step schemes allow for parallelizing in processor communication, too. Whichever variables we parallelize, we want a parallelization scheme that scales well with both particles and processors.

3 Basic Parallelization Strategies

We consider three basic schemes: full replication (parallelization strictly in particles), full domain decomposition (parallelization strictly in space), and general domain decomposition/replication (parallelization in both particles and space).

3.1 Full Replication

The traditional way to make a Monte Carlo code parallel is to repeat the mesh on multiple processors, as shown in Figure 2. We refer to this scheme as full replication. After the mesh is copied to each processor (or each processor has access to the entire mesh) the particles are split up between the processors and run with processor-dependent random number streams. Since particles are independent of each other, the processors need not communicate with each other during particle tracking. After all the particles on all the processors have finished tracking, tallies and other results are accumulated from all processors. Full replication is a simple and tremendously efficient parallelization scheme and is why Monte Carlo is sometimes called "embarrassingly parallel."

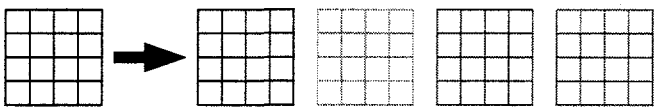


Fig. 2. Full replication, where the whole mesh is copied to each of the four processors. Full replication is a strict parallelization in particles.

3.2 Full Domain Decomposition

Unfortunately, the resolution and size of some problems produce a mesh too big to fit one processor. For these problems, we are constrained to parallelize in space (cells), where a given processor will only hold a portion of the mesh. Since cells are not typically independent of each other, in-cycle communication between processors occurs during particle tracking.

As shown in Figure 3, full domain decomposition refers to parallelizing strictly in space, where each processor holds a unique and exclusive portion of the mesh. None of the cells in the problem are replicated on multiple processors. During tracking, when a particle wants to enter a cell that is not on the processor it is on, the particle has to be sent to the processor that contains the cell. For a given mesh, the cost of this in-cycle communication between processors generally increases with both the number of particles and the number of domains.

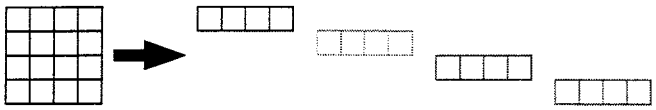


Fig. 3. Full domain decomposition, where the mesh is broken up between all four processors. Full domain decomposition is a strict parallelization in space, or cells.

3.3 General Domain Decomposition/Replication

The general domain decomposition/replication scheme replicates the mesh within the constraint of insufficient processor memory. With this general scheme, we attempt to put as much of the mesh on each processor and to replicate cells as much as possible or necessary, as shown in Figure 4. During transport, a particle would not leave its processor until necessary. The general domain decomposition/replication scheme limits to full domain decomposition as processor capacity decreases, and it limits to full replication as processor capacity increases.

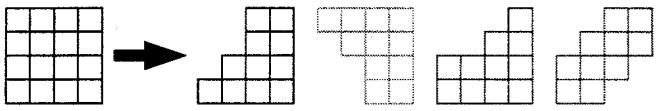


Fig. 4. The General Domain Decomposition/Replication scheme replicates the mesh as much as possible and necessary.

Time-explicit load balancing in the general scheme calls for replicating a cell based on the number of source particles it contains. A more sophisticated and time-implicit load balancing scheme takes into account how optically close a cell is to the source particles.

3.4 One-Dimensional Hypothetical Examples

To compare the different parallelization schemes, let us consider three different one-dimensional problems,

- Marshak Wave, where the wave has propagated just shy of 2 cm into the slab
- cosine temperature distribution,
- flat temperature distribution,

where the slab is divided into 8 cells, each 1 cm thick. The Marshak and cosine problems are shown in Figure 5.

For each problem, we assume that we have four processors at our disposal, but, unfortunately, each processor can hold no more than 4 cells. Thus, full replication is not an option. Overall speedup comes from raw parallel speedup minus

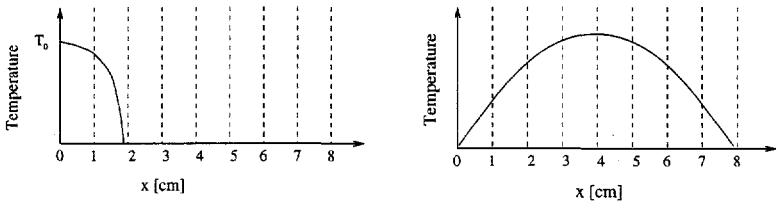


Fig. 5. Left: A Marshak Wave that has propagated almost 2 cm into an 8 cm slab. Right: A Cosine distribution of temperature.

pre-cycle, in-cycle, and post-cycle communication costs. The raw speedup in the full domain decomposition scheme comes from its space-parallel approach. The raw speedup in the general domain decomposition/replication comes from parallelization in both space and particles. The full domain decomposition topology (distribution of cells among the processors) is the same for all three problems and is shown in Fig. 6. The shaded cells are those that the processor actually contains. The general domain decomposition/replication scheme produces the topologies shown in Fig. 7.

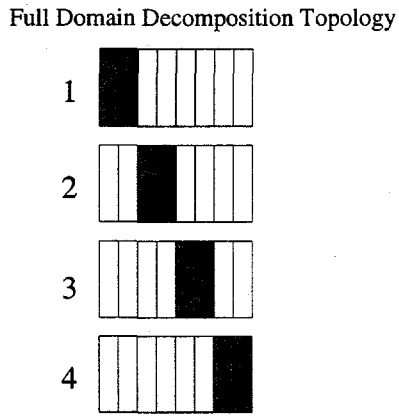


Fig. 6. The topology for the full domain decomposition is the same for all three problems.

The first problem, a Marshak Wave [3], has a steady-state, isotropic source of photons impinging upon a slab of cold material. The photons are distributed according to a Planckian at temperature T_0. The incoming photons propagate through the slab and heat the material, which, in turn, gives off more photons. Full domain decomposition will achieve serial results at best. Apart from pre- and post-cycle communication, the general strategy will achieve a raw speedup of four. Both methods require pre-cycle overhead to decompose the domain, but

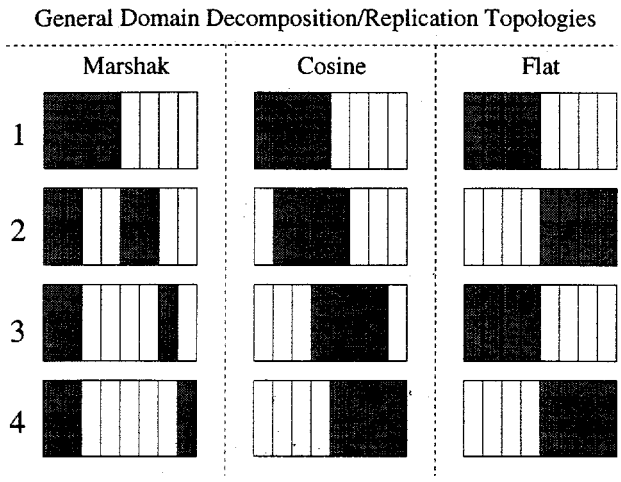


Fig. 7. The topologies for the general domain decomposition/replication scheme for the Marshak, Cosine, and Flat hypothetical problems.

the general scheme will incur larger pre-cycle overhead costs because calculating the unique domain and boundary for each processor is more expensive.

Consider the second problem, where, instead of a Marshak Wave, our one-dimensional slab has an external source in the middle with a subsequent cosine distribution of temperature. Particle activity is higher in the middle of the slab and lower near the edges. The amount of in-cycle communication will be higher if domain boundaries are placed where particle activity is high. Compared to the full domain decomposition scheme, which has no replication, the general scheme replicates the interior cells more than the outer colder cells. The price of this replication is increased communication, since each processor can talk to all the other processors. For instance, if a particle on processor 1 needs to go from the third cell to the fourth cell, it could go to processor 2, 3, or 4. Notice, though, that some of the domain boundaries in the general scheme are located where particle activity is lower. Conceivably, redundant communication could be limited to reduce the number of communication channels.

Let us now consider the third problem, where the temperature and, hence, particle activity are constant throughout the eight-celled slab. Here, the full domain decomposition scheme produces the same topology as before, while the general scheme produces alternating halves of the mesh. Full domain decomposition, due solely to parallelization in space, achieves a raw speedup of 4 since each cell has the same amount of work to do. The general scheme, due to parallelization in both space and particles, also obtains the full raw speedup of 4, since each processor has half the work to do and the mesh on each processor is replicated twice. For the full domain decomposition scheme, in-cycle communication must occur between neighboring processors, giving a total of 6 communication channels (back and forth across a domain counts as two communication channels).

For the general scheme, processor 1 can talk to processors 2 and 4, processor 2 to 1 and 3, and so on, for a total of 8 communication channels. If the problem had a large number of cells, the number of communication channels in the full domain decomposition scheme scales with the number of processors, P, as $(2P-2)$ for this one-dimensional problem. The number of communication channels in the general scheme scales as $P^2/2$, which means the general scheme's in-cycle communication does not scale well at all with the number of processors. (Note that, in the general scheme, processors 1 and 2 could be forced to talk only to each other, and likewise processors 3 and 4, for 2 send-channels replicated twice.)

So, for the basic schemes, full replication is the best scheme if the entire mesh will fit on a single processor or each processor can get access to the entire mesh. Otherwise, for problems with unevenly distributed particle activity, i.e. hot spots, the general domain decomposition/replication scheme provides more replication than the full domain decomposition scheme at the price of increased in-cycle communication. Relative payoffs from the two schemes will depend on the ratio of work required in the cells to the cost of communication across domain boundaries.

4 Two-Step Parallelization Strategies

An advanced type of parallelization strategy is a two-step scheme where the entire mesh is represented on a small set of processors, and then the small set is replicated to the rest of the processors. So each processor subset is one replicate of the mesh. Lawrence Livermore National Laboratory (LLNL) proposed a two-step scheme, which is what our two-step scheme is based upon.

Lawrence Livermore National Laboratory proposed a two-step scheme [4] for P processors, as shown in Figure 8. First, the entire mesh is fully domain decomposed on a subset of P_{set} processors. Second, the subset is replicated on the remaining $S-1$ subsets of processors, where $S = P/P_{set}$ is the number of processor subsets. In-cycle communication between processors is limited to within a processor subset. On each subset, LLNL's two-step scheme has the same qualities as full domain decomposition. However, all the work *and communication* occurring on a subset is replicated a total of S times. In other words, for the price of one subset of processors (plus pre- and post-cycle overhead), they get the work of S subsets. Jim Rathkopf, of LLNL, apparently had suggested a modification to include a small overlap of domains to handle those particles that jump back and forth across the domain boundaries [5].

Our two-step scheme, shown in Figure 9, is based on LLNL's step scheme. However, our first step consists of a general domain decomposition/replication of the entire mesh on a subset of P_{set} processors. This subset is replicated $S-1$ more times. Again, communication is limited to within a processor subset. The advantage of our two-step scheme is that "hot" cells may potentially be replicated on all P processors. Furthermore, the poorly scaling high cost of full processor cross-talk is limited to P_{set} processors.

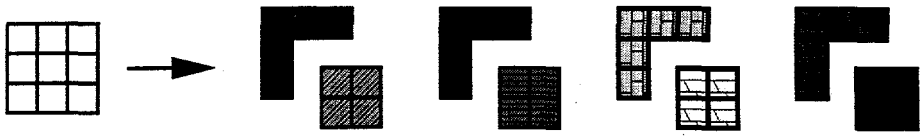


Fig. 8. LLNL's two-step parallelization scheme, where one subset of two processors simulates a single processor in the Full Replication scheme. Each processor subset uses Full Domain Decomposition to represent the whole mesh.

Fig. 9. Our two-step parallelization scheme, where one subset of processors simulates a single processor in the Full Replication scheme. Each processor subset uses General Domain Decomposition/Replication to represent the whole mesh.

5 Conclusion

We have presented a scheme for parallelizing IMC or any other Monte Carlo calculation. Full replication is the easiest way to parallelize IMC, but the meshes of the some problems are so large they will not fit on each processor. Our scheme is to use General Domain Decomposition/Replication on a subset of processors, where as much of the mesh as possible and necessary is put on each processor in the subset. The topology of the subset is then replicated on other subsets of processors. Our scheme is based upon a scheme proposed by the Lawrence Livermore National Laboratory. For both IMC alone and coupled hydrodynamics-IMC, our scheme incurs more pre-cycle communication, but it has the advantage of allowing hotter cells to be replicated on all the processors. Whether the localized full replication overcomes the extra pre-cycle communication is problem-dependent. For instance, if cells are optically thick, they require more work than the communication across domain boundaries require, and our scheme may be faster overall.

References

1. J. A. Fleck 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.
2. Geoffrey Furnish, Contain-Free Numerical Algorithms in C++ , to be published in Computers in Physics, May–June, 1998.
3. A. G. Petschek, R. E. Williamson, and J. K. Wooten, Jr., The penetration of radiation with constant driving temperature, Technical Report. LAMS–2421, Los Alamos Scientific Laboratory, 1960.

4. D. Miller and R. Procassini, private communication, Lawrence Livermore National Laboratory, 1997.
5. J. Rathkopf, private communication, Lawrence Livermore National Laboratory, 1998.