M&C 2017 - International Conference on Mathematics & Computational Methods Applied to Nuclear Science & Engineering, Jeju, Korea, April 16-20, 2017, on USB (2017)

Domain Decomposed Parallel Implicit Monte Carlo with the Data Server Model

Alex R. Long,*

*CCS-2, Los Alamos National Laboratory, P.O. Box 1663, Los Alamos, NM
along@lanl.gov

Abstract - *The large computational cost of IMC for radiative transfer is related to two factors: first, using an energy-based source strategy makes it difficult to load balance a dynamic problem. Second, domain decomposed IMC requires a large amount of parallel communication. Currently, that communication is passing particles as they move between spatial sub-domains. A mini-app, BRANSON, has been developed to study both the standard method and an alternative parallel algorithm for IMC that passes mesh and physical information between parallel processes instead of passing particles. This approach handles load balancing by dividing particles evenly between parallel processes and sending mesh data when required. The two methods are compared on simple problems. For load balanced problems, the mesh passing method has slightly improved performance compared to particle passing. For imbalanced problems, the mesh passing method strong scales with 84% efficiency with 2048 processors and simple particle passing cannot scale.*

I. INTRODUCTION

The Implicit Monte Carlo (IMC) method is a standard method for simulating thermal radiative transfer (TRT) in multiphysics problems [1]. It is also computationally expensive and can account for 80% of the runtime in HEDP problems. IMC is used to solve the thermal radiative transfer equations:

$$\frac{1}{c}\frac{\partial I}{\partial t} + \Omega \cdot \nabla I + \sigma_{\mathrm{a}} I = \sigma_{\mathrm{a}} B, \tag{1}$$

$$\frac{\partial U_{\mathrm{m}}}{\partial t} = \int_0^{\infty}\int_0^{4\pi} \sigma_{\mathrm{a}} I \, d\Omega dv - \int_0^{\infty} \sigma_{\mathrm{a}} B \, dv + S_{\mathrm{m}}. \tag{2}$$

These equations are linearized and then Eq. (1), the transport equation, is solved by inverting the streaming and collision operator with particle histories. These particle histories represent the radiative energy in the problem and how that energy moves through the material. If an energy-based source strategy is used—which is desirable for variance reduction purposes—all simulated particles represent the same amount of energy:

$$E_{particle} = E_{total}/N_{total} \tag{3}$$

With emission energy proportional to T^4, small spatial gradients in temperature can lead to large gradients in IMC particle density when the energy-based source strategy is used. Large discrepancies in particle counts across space directly translate into a load imbalance in parallel, domain decomposed IMC simulations. If the load is relatively balanced, methods exist for efficiently passing particle between parallel processes [2] [3].

Currently, two approaches exist for dealing with this load imbalance in parallel Monte Carlo transport. The first approach replicates load imbalanced sub domains [4]. The second approach is to apply a user-specified importance function to counteract the effect of temperature gradients on IMC particle density. The first approach relies on an accurate work estimate and also could potentially be imbalanced when many high particle densities travel to unreplicated spatial domains. The second approach is undesirable because it requires problem specific knowledge and changes a variance reduction tool (importance sampling) into a crude acceleration technique.

Parameter	Symbol
Batch size	n_b
Chunk size	c_{size}
Non-local mesh size	m_{size}

TABLE I. Parameters in the mesh passing algorithm

II. THEORY

To simplify the technique for load-balancing, we use the idea of data servers found in Monte Carlo neutronics [5]. When a particle requires mesh data for transport that is not local to its parallel process, the parallel process requests that mesh data from the owner of the mesh data. In contrast, the standard particle passing method sends particles to2 the parallel process that owns the non-local mesh data. If the data server model is used, the particle work can be divided up equally between all parallel process and each process can independently request the mesh data needed by its particles, wherever they might be located. We call this method "mesh passing" to distinguish it from the standard domain decomposed method of particle passing. Mesh passing performance in IMC requires effectively communicating and managing non-local mesh data, having sufficient particle work and selecting algorithm specific parameters that are appropriate for the problem and the computer architecture. A full list of parameters used in the mesh passing method is shown in Table I. Specific parameter settings for optimal performance are not evaluated here, instead a simple implementation of the method is compared to the particle passing method for a load balanced and load imbalanced problem.

The mesh passing method was implemented two ways: with MPI's standard two-sided messaging and MPI one-sided messages using remote memory access (RMA) operations and MPI shared memory windows [6]. Going forward, the abstract idea of a "parallel process" will be replaced with "MPI rank."

For all test problems an MPI rank was mapped to a single core in a multiprocessor.

1. Algorithm Parameters

The parameters that affect both two-sided MPI and one-sided MPI mesh passing method are related to memory limitations, parallel architecture and asynchronous work. The first parameter is the batch size, n_b. The batch size is the number of particles to run before processing parallel messages. If the batch size is too large, particles that need remote data will wait longer on average. If the batch size is too small the time spent checking message requests could become large enough to significantly affect runtime.

The next major parameters is the chunk size, c_{size}. Because particles that move into a given mesh cell are likely to move into neighboring mesh cells, mesh cells are grouped into spatially contiguous and memory contiguous"chunks" within a domain. When one cell is requested from a chunk of cells, the whole chunk is sent. A larger chunk size prefetches more mesh data, decreasing the number of MPI messages but increasing the amount of data sent in those messages (some prefetched cell data will likely not be used). If the chunk size it too small and a rank requires a large amount of remote data, MPI message latency could significantly affect runtime. These chunks of data are created by calling Metis on local mesh data and asking for n_{chunk} partitions, where n_{chunk} is:

$$n_{chunk} = \frac{n_{cell}}{c_{size}} \tag{4}$$

The final major parameter is the non-local mesh size, m_{size}. When remote mesh data is received, it is placed into an unordered map object that uses the global cell index as the key and the cell data as the value. This non-local mesh could grow to encompass the entire mesh, effectively replicating the problem geometry. For larger meshes, the non-local mesh must be limited to avoid running out of memory. Ideally the non-local mesh would be managed like a computer cache with a cache replacement policy, thereby saving memory that is used often (likely mesh data that is geometrically close to the local mesh). A cache replacement policy is not currently implemented and simple replacement is used instead.

There are a few minor parameters in the mesh-passing algorithm that, as yet, have not had a significant effect on runtime. The number of concurrent sends and receives of mesh data a rank may have is limited to not overwhelm the MPI library.

2. Implementation

Mesh passing and particle passing were implemented in the IMC code Branson. Branson is a small, lightweight IMC code meant to test parallel methods. It implements both simple particle passing from [2] and mesh passing. Branson has only temperature dependent, gray opacities and simple cartesian meshes. It does not comb census particles or have any acceleration techniques for particles in diffuse regions. The lack of a multigroup treatment keeps regions from being both optically thick and thin to different groups. That case would be valuable to assess and is a weakness in Branson as a proxy application for IMC. Combing and acceleration schemes simplify load balance and memory demands in IMC so Branson represents a more conservative case for these issues.

The two-sided MPI implementation checks for mesh cell requests after transporting n_b particle histories. The rank then sends all of the requested cells at once to the requesting MPI rank. It then checks for received cell data needed by its particles and adds any received data to the non-local mesh. When new non-local mesh is received, particles that are waiting for remote data are transported as far as possible. When a rank completes its particles it sends a completion message up the binary tree (as in [2]) and continues servicing requests for its mesh data.

The one-sided messaging implementation allocates mesh data in an MPI window. This window can then be read by other ranks. A passive, one-sided request is made by the rank that requires remote mesh data. These RMA operations are done at a low level and optimally do not involve the operating system of the target MPI rank. The requesting rank tests these requests after running b_s particles and upon completion transports particles that are waiting for mesh data. The mesh data is requested in size of c_{size}. Because operations are one-sided, each MPI rank can complete its particle work independently and does not need to send or check for completion messages. This makes the parallel algorithm simpler compared to particle passing and two-sided MPI mesh passing.

Both mesh-passing methods have a copy of the absorbed energy tally for the entire mesh and an `MPI_Allreduce` is called at the end of a timestep to get the correct absorbed energy tally. This is a known weakness of the mesh passing method and must be addressed before it can be recommended for adoption.

3. Load balancing

Load balancing in replicated domain-decomposed Monte Carlo applications involves finding spatial domains (all the mesh data on an MPI rank, for instance) with a large amount of particle work and replicating those domains based on functions that estimates the cost of completing the work in that domain. In the mesh passing algorithm, load balancing is done at the particle level—particle work is divided evenly across the processors. Particle work can be specified as a "work packet", which is some number of particles to make in a given cell with a given energy. Particle work can also be a literal particle, as in the census. Particle work specified as a work packet avoids storing and communicating full particle data, which could easily consume the whole memory of a node in load imbalanced problems. To limit communication of cell and particle data, a process with excess work will preferentially send emission work before census work—emission particles are born throughout the timestep and thus will likely travel a shorter distance in a timestep and require less remote data compared to census particles. To allow for overlap of parallel and serial work, work from remote ranks is placed at the top of the particle stack, meaning that requests for remote data will be made early in the simulation.

Parameter		Value
particles	=	2.0×10^8
Δt	=	0.01 ns
t_f	=	0.1 ns
cells	=	40^3
$\Delta x, \Delta y, \Delta z$	=	1/40 cm
f	=	0.154
σ_a	=	$\frac{100}{T^3}$ cm^{-1}
σ_s	=	0.0 cm^{-1}
c_v	=	0.1 $\frac{\text{jk}}{\text{g keV}}$
ρ	=	3.0 $\frac{\text{g}}{\text{cm}^3}$
$T_{m,0}$	=	1.0 keV
$T_{r,0}$	=	1.0 keV

TABLE II. Problem description for the load balanced IMC transport problem

Parameter		Value
particles	=	1.0×10^9
Δt	=	0.01 ns
t_f	=	0.05 ns
cells	=	400^2
$\Delta x, \Delta y$	=	1/100 cm
hot corner	=	$x \in (0, 0.05$ cm), $y \in (0, 0.05$ cm)
σ_a	=	50.0 cm^{-1}
σ_s	=	0.0 cm^{-1}
c_v	=	0.1 $\frac{\text{jk}}{\text{g keV}}$
ρ	=	1.0 $\frac{\text{g}}{\text{cm}^3}$
$T_{m,cold}$	=	0.01 keV
$T_{m,hot}$	=	1.0 keV

TABLE III. Problem description for the hot corner load imbalanced IMC transport problem

III. RESULTS AND ANALYSIS

1. Load Balanced Problem

A strong scaling study was performed for a standard, load balanced problem, with the setup shown in Table II. The results are shown in Fig. (1). This problem was run on the Trinitite machine at Los Alamos National Laboratory. The particle passing method exhibits strong scaling out to 1024 cores at about 63% efficiency compared to one node (32 processors). The two sided mesh passing strong scaling efficiency at 1024 cores is about 91% and the one-sided mesh passing strong scaling efficiency at 1024 cores is about 86%. At 1024 cores, there are 2.0×10^5 particles per MPI rank, which is a small amount of work and illustrates the difficulty of strong scaling studies. The mesh-passing method exhibits better scaling than the particle passing method. This is likely because there are more ranks for the same amount of mesh data—there are effectively more servers able to handle requests for the same data. This benefit eventually fades as the processor count increases because the domain sizes are smaller and a greater fraction of particles will need non-local mesh data during transport.

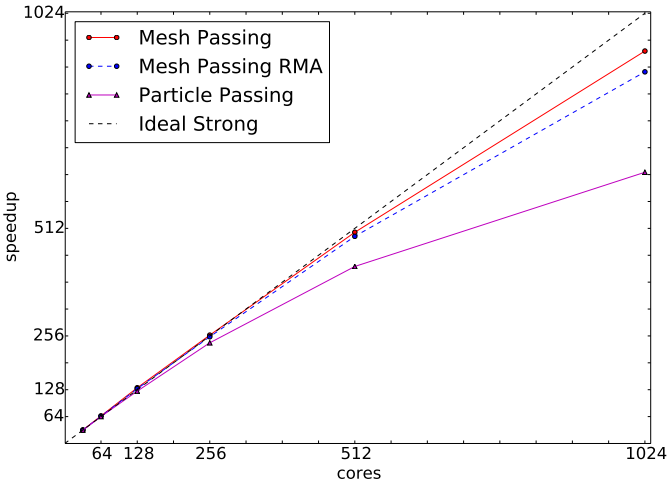

Fig. 1. Strong scaling for the load balanced problem in Table II for the particle passing and mesh passing methods

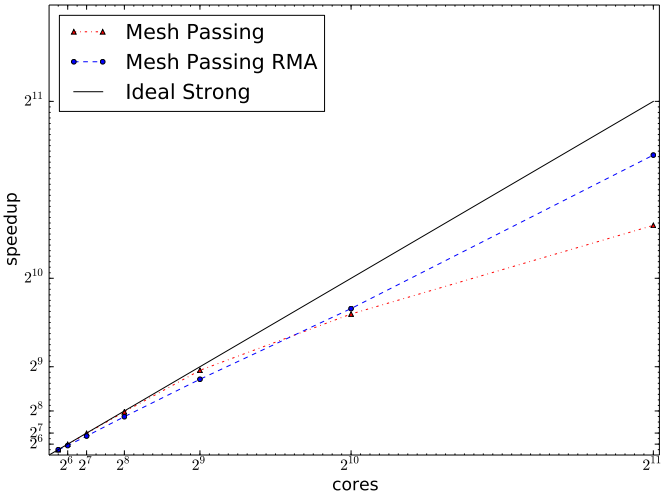

Fig. 2. Strong scaling for the hot corner problem in Table III for the one-sided and two-sided mesh passing implementations

2. Load Imbalanced Problem

The hot-corner problem shown in Table III was run with 2048 processors using two-sided and one-sided mesh passing. This problem setup puts more than 99.99% of the particles on one MPI rank. The scaling results are shown in Fig. (2) and the total transport runtimes are shown in Fig. (3). Running a smaller version of this problem (1.0×10^7 particles and 200^2 cells and 20 timesteps) on one node (32 processors) with the particle passing method takes about 4122.19 seconds and only a slight improvement is expected as more processors are added (eventually the hot corner will be spread across several MPI ranks). The runtime for the particle passing method with 32 processors is about 10% slower than the single processor runtime with particle passing. This result is expected because the single rank transporting all the particles no longer has access to all of the node resources compared to the serial run.

The two-sided mesh passing implementation achieves

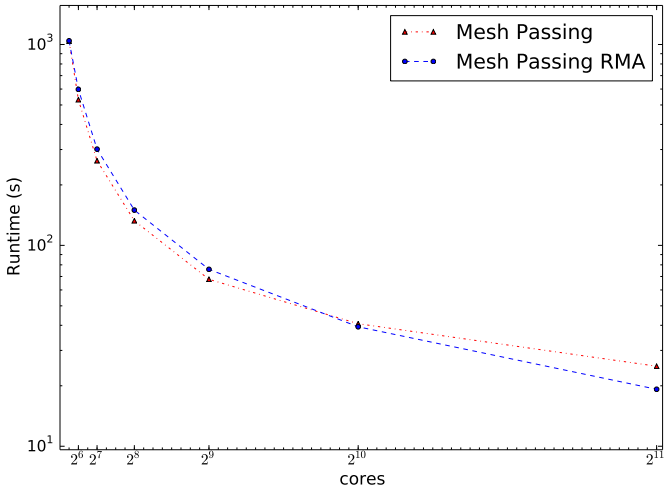

Fig. 3. Total transport runtime for the hot corner problem in Table III for the one-sided and two-sided mesh passing implementations

a speedup of about 1737 and about 84% efficiency at 2048 processors (compared to the performance of single node with 32 processors). Both implementations on a single node run in about 1040 seconds. At 2048 processors, each timestep takes less than about 4 seconds and there are about 600000 particles on each MPI rank. The runtime and scaling is slightly better in the two-sided version for core counts smaller than 1048. This could because the two-sided implementation sends all requested mesh chunks at once, as opposed to a single chunk per message in the one-sided implementation. Above 1048, the performance and scaling of the two-sided implementation decrease. This could be because at high core counts mesh sizes are smaller and can be sent all at once, eliminating the benefit of aggregating mesh requests. One-sided messaging is known to have less latency than two-sided messaging—this could explain the improved scaling compared to two-sided messaging.

In these results, a range of batch sizes and chunk sizes were used and the best result for each core count is shown in the runtime and scaling results. The non-local mesh size was kept at 5000 cells for all runs. Generally, a chunk size and batch size that lead to the least number of MPI messages give the best performance. This result suggests that the network latency and not the bandwidth is the limiting factor in performance. The results were less sensitive to these parameters are large core counts where the mesh size of each rank was relatively small.

At 2048 MPI ranks, load balance takes less than 1% of the total runtime. The current load balancing algorithm in Branson can be an all to all communication when the work is on one rank. The load balancing algorithm used by O'Brien [4], which is known to be O(log(N_p)), has also been implemented in Branson. This algorithm achieves faster load balancing but it is more difficult to give spatially collocated particle work to all processes, which incurs more parallel communication and makes the transport runtime slightly slower. This issue is being investigated because an O(log(N_p)) algorithm for load balancing will be essential at very high core counts.

IV. CONCLUSIONS

Initial results for the mesh passing method show significant improvement over standard particle passing in both load balanced and dynamic, load imbalanced IMC problems. More detailed weak scaling studies are needed to fully evaluate the mesh passing method as a potential replacement for particle-passing in parallel IMC.

The mesh passing method has several known deficiencies that could be addressed in future work. The issue of tallies on remote mesh must be addressed as replicating and reducing tally information is not practical for large meshes and high core counts. This issue could be overcome with one-sided accumulate operations. A full parameter study is needed to identify the optimal balance between parallel communication and computational work in the mesh passing method. If a multigroup treatment is used for opacities an individual cell could require much more memory and thus larger MPI messages when communicating non-local mesh data. The results point towards MPI latency being the bottleneck, which means larger messages could be sent without significantly impacting performance. The relationship between cell memory footprint and performance should be measured.

V. ACKNOWLEDGMENTS

Gabe Rockefeller provided guidance on the algorithm and its implementation. Los Alamos National Laboratory is operated by Los Alamos National Security, LLC for the U.S. Department of Energy under Contract No. DE-AC52-06NA25396.

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. T. A. BRUNNER, T. J. URBATSCH, T. M. EVANS, and N. A. GENTILE, "Comparison of four parallel algorithms for domain decomposed implicit Monte Carlo," *Journal of Computational Physics*, **212**, *2*, 527 – 539 (2006).
3. T. A. BRUNNER and P. S. BRANTLEY, "An efficient, robust, domain-decomposition algorithm for particle Monte Carlo," *Journal of Computational Physics*, **228**, *10*, 3882–3890 (2009).
4. M. J. OŠBRIEN, P. BRANTLEY, K. JOY, and F. GYGI, *Scalable Domain Decomposed Monte Carlo Particle Transport*, Ph. D. Dissertation, University of California, Davis (2014).
5. P. K. ROMANO, B. FORGET, and F. BROWN, "Towards Scalable Parallelism in Monte Carlo Particle Transport Codes Using Remote Memory Access," (2010).
6. "MPI: A Message-Passing Interface Standard Version 3.1," (June 2015).