LA-UR-04-4532

Approved for public release;
distribution is unlimited.

Title: HIGH PERFORMANCE COMPUTING & MONTE CARLO

Author(s): FORREST B. BROWN & WILLIAM R. MARTIN

Submitted to: American Nuclear Society 2004 Winter Meeting,
Washington, DC, 14-18 November 2004

High Performance Computing and Monte Carlo

Forrest B. Brown[1] and William R. Martin[2]

[1]*Diagnostics Applications Group (X-5), Los Alamos National Laboratory,*
PO Box 1663, MS F663, Los Alamos, NM, fbrown@lanl.gov
[2]*Department of Nuclear Engineering & Radiological Sciences, University of Michigan,*
Ann Arbor, MI, 48104, wrm@umich.edu

INTRODUCTION

High performance computing (HPC), used for the most demanding computational problems, has evolved from single processor custom systems in the 1960s and 1970s, to vector processors in the 1980s, to parallel processors in the 1990s, to clusters of commodity processors in the 2000s. Performance/price has increased by a factor of more than 1 million over that time, so that today's desktop PC is more powerful than yesterday's supercomputer. With the introduction of inexpensive Linux clusters and the standardization of parallel software through MPI and OpenMP, parallel computing is now widespread and available to everyone.

Monte Carlo codes for particle transport are especially well-positioned to take advantage of accessible parallel computing, due to the inherently parallel nature of the computational algorithm. We review Monte Carlo particle parallelism, including the basic algorithm, load-balancing, fault tolerance, and scaling, using MCNP5 as an example. Due to memory limitations, especially on single nodes of Linux clusters, domain decomposition has been tried, with partial success. We conclude with a new scheme, data decomposition, which holds promise for very large problems.

PARTICLE PARALLELISM

The most common approach to high-performance particle transport Monte Carlo is particle parallelism using a *master/slave algorithm*, where a master process distributes individual histories to different slave computing nodes, which then analyze the histories concurrently using one or more threads (Fig. 1). MPI message-passing is used to distribute problem data to the slaves and collect the tallied results, while OpenMP is used for threading computation within each slave. Two characteristics of these calculations are ideal for parallelism: (1) compact 3D combinatorial geometry is used, so that memory storage is not required for a large 3D mesh, and (2) individual particle histories are independent, so that slaves do not need to interchange any information with other slaves during computations. The master/slave algorithm requires that each slave node has its own copy of problem geometry, cross-section data, and tallies. These data may be shared among threads for slaves with multiple CPUs.

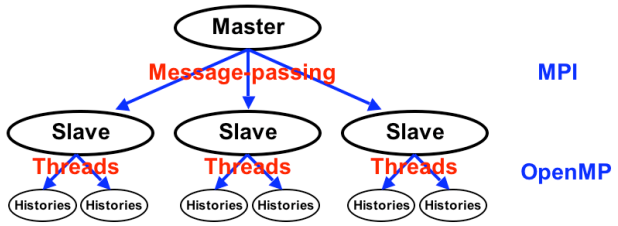

Fig. 1. MCNP5 master/slave algorithm

Load-balancing and fault tolerance must be handled robustly in HPC Monte Carlo codes. In MCNP5 [1,2], for example, fault-tolerance is handled by having the slaves periodically *rendezvous* with the master, stopping the computation of histories and sending current tally data to the master, so that the master can backup the data in a *dump-file*. If the system has a hardware failure, the most recent dump-file can be retrieved to continue the calculations. To achieve load-balancing, MCNP5 uses *self-scheduling* by the slaves (Fig. 2). The total number of histories is divided into moderate-sized chunks. When a slave is idle, it requests a new chunk of work from the master. Faster slave nodes process more chunks of work than slower nodes.

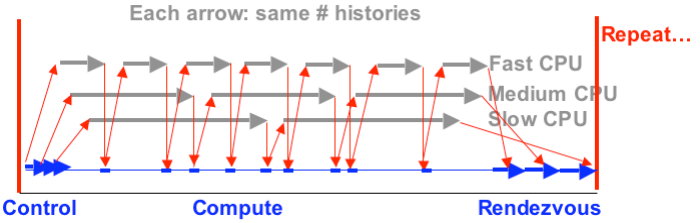

Fig. 2. MCNP5 self-scheduling and rendezvous

The particle parallelism approach used in MCNP5 (master/slave algorithm, using MPI and OpenMP, with self-scheduling and rendezvous) has been proven to be effective on a wide variety of HPC systems, including Linux clusters, Windows clusters, workstation clusters, shared-memory parallel systems, and ASC teraflop systems. Fig. 3 shows the measured speedups for MCNP5 on the Q system at LANL, a 20 TFLOP ASC system. A parallel speedup of 2600x was achieved using 3700

processors. For particle parallelism, speedups are limited by the communication and synchronization overhead required for periodic rendezvous. Models of the parallel scaling behavior are well-known and understood [3, 4]. Minimizing the rendezvous overhead results in improved parallel efficiency.

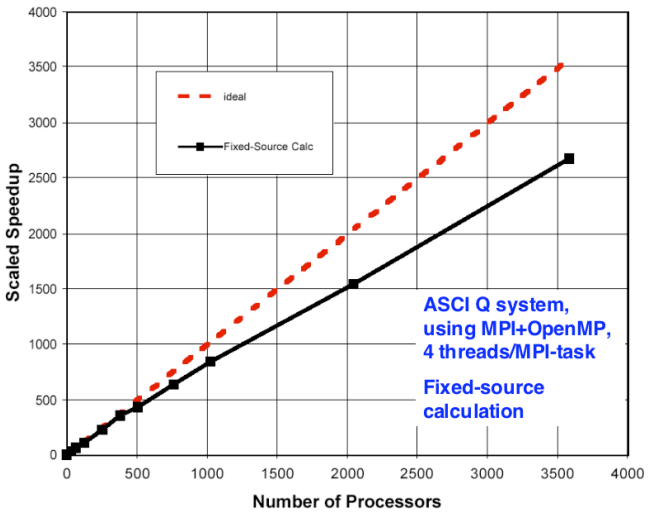

Fig. 3 MCNP5 parallel speedup on Q system

DOMAIN DECOMPOSITION

For Monte Carlo codes which use a mesh representation of geometry, some applications may require so much memory storage for the mesh and tallies that spatial domain decomposition may be necessary [5, 6]. The physical problem is partitioned into subdomains which will fit in the memory of different processors. If domain decomposition is used, then it is necessary to transfer particles between domains when they reach boundaries. These transfers result in slave-to-slave communications, synchronization delays, and severe imbalances in processor load, all of which are at odds with the natural particle parallelism. Domain-decomposed Monte Carlo methods have generally shown only moderately successful performance and scaling for reactor problems, and poor results for time-dependent problems. They have not been successful to date in scaling to many 100s or 1000s of processors.

PARTICLE PARALLELISM WITH DATA DECOMPOSITION

To overcome performance difficulties with domain decomposition, alternate approaches are being investigated for problems with very large memory requirements. With MCNP5, for example, there is much interest in performing medical physics dosimetry calculations on VIP-man, a standard tomographic model consisting of 3.7 billion voxels which represent the geometry and materials. Since the required memory storage exceeds that available on Linux cluster nodes, the voxel data must be partitioned across different nodes. Variations in the current MCNP5 particle parallelism are being investigated. The basic particle parallelism algorithm will be retained, with the addition of a set of *data nodes*. The additional data nodes act as data servers, sending blocks of voxel data to the slaves on demand. That is, rather than move particles to data locations (as for domain decomposition), the modified MCNP5 algorithm will move blocks of data to the particles as needed. Unlike domain decomposition, this approach should scale to 1000s of processors for memory-intensive applications that cannot be contained within a single processor.

CONCLUSIONS

Monte Carlo codes for particle transport are often the first production codes in use on new HPC computers. Particle parallelism is an effective and highly portable algorithm which requires little or no modification for advanced computers or new architectures.

REFERENCES

1. X-5 MONTE CARLO TEAM, “MCNP – A General Monte Carlo N-Particle Transport Code, Version 5, Volume I: Overview and Theory,” LA-UR-03-1987, Los Alamos National Laboratory (April, 2003).
2. F. B. BROWN, J. E. SWEEZY, J. T. GOORLEY, “MCNP5 Parallel Processing Workshop,” LA-UR-03-2228, Los Alamos National Laboratory (2003).
3. S. MATSURA, F. B. BROWN, R. N. BLOMQUIST, *Trans. Am. Nucl. Soc.* (Nov. 1994).
4. A. MAJUMDAR and W. R. MARTIN, “Performance Measurement of Monte Carlo Photon Transport on Parallel Machines,” *PHYSOR 2000: ANS Int. Topical Meeting*, Pittsburgh (May 2000).
5. S. R. LEE, S. D. NOLEN, F. B. BROWN, “MC++ Monte Carlo Code for ASCI, ” Los Alamos National Laboratory report (1998).
6. R. PROCASSINI, et al., “Design, Implementation and Testing of MERCURY, a Parallel Monte Carlo Transport Code,” *Proc. ANS Math. & Comp. Topical*, Gatlinburg, TN, April 6-11 (2003)