Proceedings of the Reactor Physics Asia 2015 (RPHA15) Conference
Jeju, Korea, Sept. 16-18, 2015

Hybrid Shared Memroy/Message Passing Parallel Algorithm in Reactor Monte Carlo Code RMC

Feng YANG *, Ganglin YU and Kan WANG
Department of Engineering Physics
Tsinghua University, Beijing, 100084, P.R. China
yangfeng14@mails.tsinghua.edu.cn

1. Introduction

Over the past 15 years the architectures of high-performance computing (HPC) systems have become increasingly heterogeneous with individual computing cores subdivided into nodes and even auxiliary co-processing units such as graphics processing units (GPUs) or the Intel many integrated core (MIC) co-processors. In fact, future computer platforms move toward a larger numbers of nodes and cores with lower memory available per node.

Today, several production Monte Carlo (MC) codes[1][2] use some types of hybrid parallelism strategy that makes use of shared-memory parallelism within a computing node and message-passing parallelism between nodes, which enables the codes to utilize available computing resources as much as possible, both among and within computing nodes on a cluster.

The so-called hybrid parallelization is also possible with Reactor Monte Carlo Code RMC by activating both parallel algorithms together. On HPC which typically consists of several or several-ten thousand nodes equipping 16-32 computing intra-cores per node, simulation of RMC can be carried out in the hybrid parallelization with the MPI inter-node parallelization and the OpenMP intra-node parallelization.

2. Parallel Algorithms For Monte Carlo

In this section we review the basic strategy for hybrid message-passing and shared-memory parallelism in MC particle transport codes and introduce standard terminology that will be used throughout the remainder of the paper.

2.1 Message Passing Parallelism (MPI)

During parallel calculations, multiple independent MC processes are launched simultaneously across interconnected processors in a computing system. These processes communicate with each other during the MC simulation by exchanging messages, typically via the message-passing interface (MPI) standard.

In message-passing parallelism, independent processes work together by exchanging sets of data which are abstractly referred to as "messages". This type of parallelism works especially well when a calculation can be subdivided into independent tasks and the intermediate results from these tasks can be combined to give the final solution. Parallel efficiency in message passing schemes is generally improved by maximizing the amount of time that processes can work independently and minimizing the number and frequency of messages sent between processes.

Particle-history parallelism with message-passing is a natural fit for MC neutral-particle transport because each particle history can be run independently from all other particle histories. As a result, particle transport routines such as cross section lookups, collision processing, ray tracing, and tally scoring can be isolated to individual processes, avoiding the need for interprocess communication by these routines.

In general, communication between parallel MC processes is only required for problem initialization, input/output operations, and periodic synchronization of the processes.

2.2 Shared Memory Parallelism (OpenMP)

Message passing parallelism works extremely well for calculations that use less than the "fair-share" process-memory allocation on a compute node, which is defined as the total memory for a node divided by the number of compute cores on the node. The fair-share memory allocation defines the maximum model size that can be fully domain-replicated on all cores of the node without swapping.

Model sizes for full-core 3D commercial nuclear reactor benchmark calculations now routinely exceed the fair-share memory allocation on contemporary HPC clusters, ~4 – 8 GB of memory per core. To perform these large-memory calculations would require leaving some compute cores idle on each node in order to boost the available memory per-processor, leading to inefficient utilization of HPC resources.

Domain decomposition[3] and shared-memory parallelism[4] are both potential solutions to the memory limitations created by the use of full domain-replication in parallel calculations, and both techniques have been employed successfully by production-level MC codes.

However, in many situations shared-memory parallelism offers several benefits over domain decomposition. First, a shared memory parallelism approach using OpenMP directives is typically easier and less disruptive to backfit into existing code architecture. Secondly, current HPC trends continue to

point towards an increase in both the number of cores and total memory per compute node, but an overall decrease in fair-share memory allocation per core. Finally, the OpenMP Architecture Review Board continues to signal an intention to provide native support for accelerator and coprocessor devices as a part of the OpenMP 4.0 standard. Thus, implementation of shared-memory parallelism using OpenMP offers the potential for simplified co-processor support in the future.

2.3 Hybrid MPI/OpenMP Parallelism

In RMC, two choices of parallel computing functions are available. One is the parallel computing using the message passing interface (MPI) protocol and the other is the parallel computing using the open multi-processing (OpenMP) directives

The MPI parallelization is a type of distributed-memory style where the memory space for each core is maintained independently while the OpenMP parallelization is a type of shared-memory style where a part of memory space can be shared among cores within a single computer or a single node of supercomputers.

3. Several Parallel Algorithms

First the consistency between serial calculation mode and different parallel calculation modes must be guaranteed.

Second the new parallel mode should have a high efficiency. The key to maximizing parallel efficiency in a MC code is to design the code to minimize the amount of inter-process communication during parallel simulations.

3.1 Random Number Generation

For pseudo-random number generation, most MC codes use a standard linear congruential generator (LCG), often based on the algorithm and parameters available in Ref. [5].

One random number stream is only depend on the random seed when the LCG's parameters are determined. In MPI parallelelism communication between parallel MC processes are excuted to distribute random seeds to each process. In OpenMP parallelism the each for loop is excuted out of order. To maintain reproducibility for criticality problems for parallel calculations, the random seeds which calculated before the for loop are stored in the fission bank.

Using the above two algorithms, the consistency between serial calculation mode and different parallel calculation modes is guaranteed.

3.2 Fission Bank Reordering

During a criticality calculation, neutrons created in fission are stored in the "fission bank" to be used as the starting source for the next iteration of the calculation. Using a single computational thread, the neutrons in the fission bank will be stored in a particular, well-defined order (i.e., successive fission sites from the first neutron, then the second, etc.). Using OpenMP threading and/or MPI message-passing parallelism, the ordering of fission sites within the fission bank is indeterminate.

To maintain reproducibility for criticality problems for parallel calculations, the fission-bank needs to be reordered at the end of each cycle into a unique ordering that is independent of the number of threads or MPI processes.

4. Parallel Scaling and Performance

To study the parallel scaling and efficiency of RMC, including the hybrid parallelism strategy and parallel algorithms described in the previous sections, Hoogenboom-Martin[6] (H-M) and BEAVRS benchmark [7] are selected. The reactor core arrangement are shown in figure 1. The two studies are performed on Intel Xeon CPU E5-2620 v2 @2.10GHz Windows 64bit platform.

The H-M reactor critical benchmark was release d by Hoogenboom et al. in 2011 with 60 nuclides i n the fuel region.

The BEAVRS Benchmark was released by MIT in 2013 to provide data from an operating nuclear reactor to the public to allow for validation of methods developments. This benchmark is similar to the data provided in the VERA Progression Benchmark.

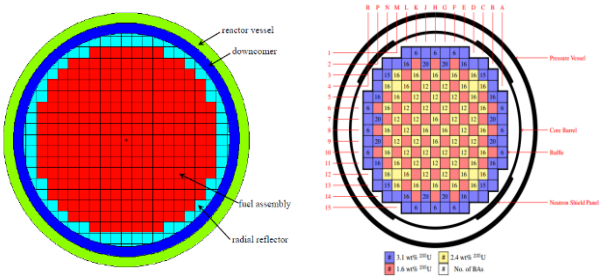

Fig. 1. H-M benchmark and BEAVRS benchmark

4.1 Shared Memory and Message Passing Parallelism Scaling

Table I shows the consistency between serial ca lculation and different parallel calculation modes.

Table I. Results for the consistency between serial and different parallel calculation modes

Benchmark	Calculation Mode	k-effective
H-M	Serial Mode	1.012962±0.001892
	MPI Mode	1.012962±0.001892
	OpenMP Mode	1.012962±0.001892
	Hybrid Mode	1.012962±0.001892
BEAVRS	Serial Mode	1.027068±0.002515
	MPI Mode	1.027068±0.002515
	OpenMP Mode	1.027068±0.002515
	Hybrid Mode	1.027068±0.002515

In Table I the random number generation and fissi on bank recording algorithms are verified effective.

4.2 Shared Memory and Message Passing Parallelism Scaling

In different parallel modes, speedup and memory cost are cared most. Because of data race and loca l acceleration, ideally it is expected the speedup in shared-memory parallelism is nearly same as MPI p arallelism and the memory cost is cut down.

The speedup in H-M and BEAVRS benchmarks are displayed in figure 2 and figure 4. The memory cost are expressed in figure 3 and figure 5.

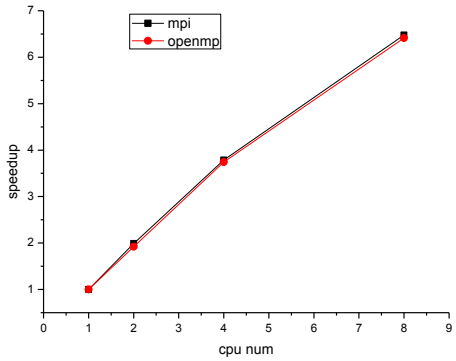

Fig. 2. Speedup of H-M benchmark

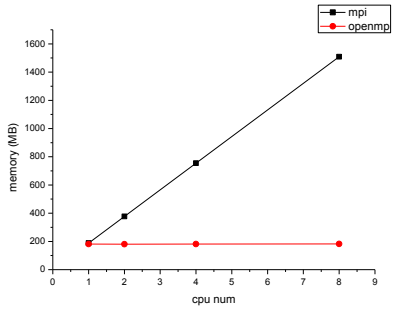

Fig. 3. Total memory cost of H-M benchmark

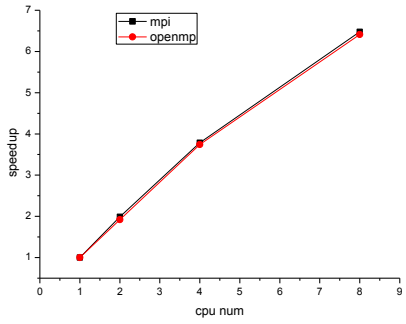

Fig. 4. Speedup of BEAVRS benchmark

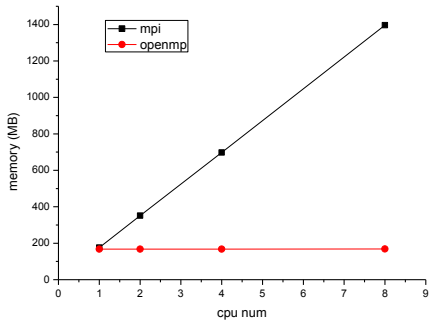

Fig. 5. Total memory cost of BEAVRS benchmark

From the figure 2-5, we can conclude that the shared-memory parallel implementation in RMC has a perfect performance in both speedup and memory cost.

4.3 Hybrid OpenMP and MPI Parallelism Scaling

In order to verify the effectiveness of hybrid O penMP and MPI parallelism scaling, BEAVRS bench mark is selected. Figure 6 shows the speedup in hy brid parallel mode and figure 7 display the memory cost in different processes\threads.

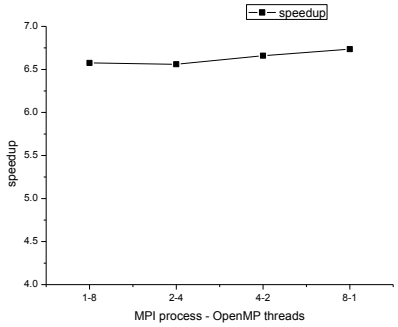

Fig. 6. Speedup of BEAVRS benchmark in hybrid p arallel

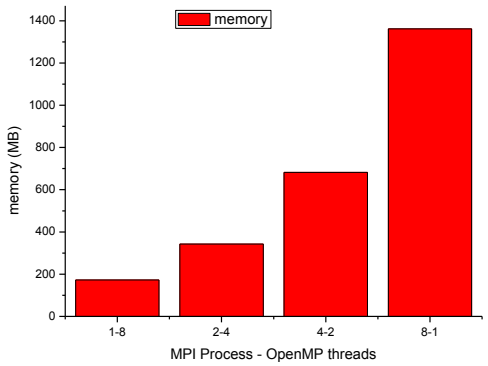

Fig. 7. Total memory cost of BEAVRS benchmark i n hybrid parallel

Through the figure 6 and figure 7, we find tha

t the parallelism efficiency are between 82% and 8 4%. The memory cost is cut down from ~1.4GB t o ~200MB. The study's target is reached in the pr eliminary results.

5. Conclusions

Keeping up with the continuing trend for increases in the number of computing resources within modern HPC systems is a significant challenge for teams that develop and maintain high-performance MC radiation transport solvers. In order to address this challenge, RMC has adopted a hybrid message-passing and shared-memory approach to parallelism, which enables the code to utilize available computing resources, both among and within computing nodes on a cluster. Shared-memory parallelism is especially valuable for large-model simulations because it enables multiple compute cores within a node to share common model information such as the geometry description, material and composition definitions, and cross section data, thus reducing the memory usage per processor.

This paper provides details on two novel parallel algorithms for processes that play a fundamental role in MC radiation transport simulations: random number generation and fission bank recording.

The parallel performance has been studied by using two complex and representative benchmarks, H-M and BEAVRS benchmark. The preliminary results show the nearly ideal speedup in hybrid parallelism and the huge memory savings.

In future work, we shall focus on the massively parallel burnup calculation and the scalability up to millions of core.

References

1. Griesheimer D P, Gill D F, Nease B R, et al. MC21 v. 6.0–A continuous-energy Monte Carlo particle transport code with integrated reactor feedback capabilities[J]. Annals of Nuclear Energy, 2014.
2. Francois-Xavier Hugot, et al. HIGH PERFORMANCE MONTE CARLO COMPUTING WITH TRIPOLI® : PRESENT AND FUTURE. ANS MC2015 — Joint International Conference on Mathematics and Computation (M&C), Supercomputing in Nuclear Applications (SNA) and the Monte Carlo (MC) Method, Nashville, Tennessee · April 19–23, 2015.
3. Romano P K. Parallel algorithms for Monte Carlo particle transport simulation on exascale computing architectures[D]. Massachusetts Institute of Technology, 2013.
4. Siegel A R, Smith K, Romano P K, et al. Multi-core performance studies of a Monte Carlo neutron transport code[J]. International Journal of High Performance Computing Applications, 2014, 28(1): 87-96.
5. L'ecuyer P. Tables of linear congruential generators of different sizes and good lattice structure. Mathematics of Computation of the American Mathematical Society, 1999, 68(225): 249-260.
6. Hoogenboom J E, Martin W R. A proposal for a benchmark to monitor the performance of detailed Monte Carlo calculation of power densities in a full size reactor core. Ann Arbor, 2009, 1001: 48109-2104.
7. Horelik, N., et al. "Benchmark for Evaluation and Validation of Reactor Simulations (BEAVRS), v1. 0.1." Proc. Int. Conf. Mathematics and Computational Methods Applied to Nuc. Sci. & Eng. 2013.