Research on Parallel Discrete Event Simulator based on a CPU+MIC Platform

Huilong Chen[1], Yiping Yao[1], Tianlin Li[1], Jin Li[1]

[1]College of Information System and Management, National University of Defense Technology, Changsha 410073, China

[2]State Key Laboratory of High Performance Computing, National University of Defense Technology, Changsha 410073, China

email: chenhuilong@nudt.edu.cn

Keywords: PDES simulator, MIC, Many Integrated Core, many-core coprocessor

Abstract. The widespread of many-core processors brings new opportunities to enhance performance of PDES applications. MIC is a relative new many-core architecture compared with the widely-used GPU. In our previous paper, experiments for evaluating performance of a PDES simulator ROSS are designed and tested on MIC and CPU respectively. However, no related works have been done on evaluation PDES simulators on CPU+MIC collaborative platform. We observe that the MPI based ROSS performs poorly, while the multi-thread ROSS is not able to work on CPU+MIC platform. We propose a PDES simulator based on a MPI+OpenMP hybrid parallel modeled ROSS which can be called as ROSS-OMPI. We use MPI to handle the interactive events between CPU and MIC, while several OpenMP threads are issued to process events in parallel inside CPU and MIC respectively. The experiment results show that the hybrid model of ROSS brings better performance.

Introduction

During the last few decades, with the deeper use of modeling and simulation, more and more large-scale complex systems have been introduced into this field. As a consequence, the execution of simulation programs has become increasingly time-consuming. Parallel Discrete Events Simulation (PDES) is considered as a feasible solution to accelerate simulation execution [1], which is able to exploit the parallelism of simulation systems to improve the performance of simulation applications with parallel processing. In this case, High-performance computing (HPC) platforms and well-designed parallel simulation engines or simulators are necessary for PDES to obtain high performance. In recent years, with the widespread use of multi-core processors, the HPC platforms have been evolved from symmetric multiprocessor machines (SMP) or cluster systems to multi-core cluster systems. At the same time, intensified investigation has been conducted on evaluation and improvements of PDES simulators based on multi-core cluster systems [2]-[6]. At present, with the emergence and successful use of many-core processors, such as GPU, etc., HPC platforms are developing fast in the direction of heterogeneous architecture composed of both multi-core processors and many-core coprocessors.

Generally speaking, one of the characteristics of heterogeneous architecture with multi-core and many-core is the utilization of many-core coprocessors based on multi-core host machines. GPU, Tilera and MIC etc. are among the relatively common used many-core processors. Currently, GPU is a widely-used many-core coprocessor in the field of HPC. As a matter of fact, computing platforms based on hybrid architecture of multi-core CPU and GPU has long been ranked top-ten in the domain of super computers, cases in point are the once top-one super computers Tianhe-1A and Titan. Meanwhile, there have been many excellent researches on performance analysis and optimization of PDES on GPU [7]-[11]. For another many-core processor, Tilera, which is mainly applied to embedded devices in the area of network and multi-media, is said to possess considerable parallel processing ability. Jagtap [12] has done research on the conformation and development of PDES simulator on Tilera. As a general purpose investigation, Zhang [13] used a “simulation of

simulation" method to study the development space of many-core platforms applied to PDES, which is counted to be a prophase of the application of general many-core computing platforms in the PDES field.

MIC (Many Integrated Core) is a new many-core architecture brought out by Intel. Nowadays, it has been used in the super computers such as Tianhe-2 and Stampede. It is remarkable to mention that Tianhe-2 has been continuously holding a top-one position in the world super-computer in the last two years. Saini [14] has introduced the characteristics of MIC in detail. Besides, he tested the memory bandwidth and latency of MIC, the performance of MPI, OpenMP functions and the I/O units with the help of several classical micro-benchmarks. Performance for actual applications of MIC has been tested and analyzed by some classical science and engineering applications as well. Nevertheless, there are few studies on the performance evaluation of PDES simulators based on MIC platforms (so far as we know from the published information). Child [15] did research on the SCC many-core platform, which is considered as the predecessor of MIC, to explore the performance of PDES simulators.

As a relatively new many-core architecture processor, MIC has presented formidable computing potential. Furthermore, benefiting from using classical x86 architecture cores, MIC coprocessor is compatible with almost all programs designed for general purpose CPUs. In other words, the code developed for general purpose CPUs is able to run on MIC only by recompiling. In our previous paper [16], we tested the performance of a classical PDES simulator ROSS on CPU and MIC respectively. The results showed that the multi-thread version ROSS performs much better than MPI based ROSS on MIC. Furthermore, parallelizing and vectorizing the code of heavy computation models in PDES application will enhance the total performance significantly on MIC. However, the previous work does not consider performance of ROSS on the CPU+MIC collaborative computing platform. On one hand, though ROSS-MT performs quite well on both CPU and MIC respectively, an obstacle exists when we need transfer messages between CPU and MIC, as the limitation of data transferring mechanism with offload mode of MIC. Also, the communication speed between CPU and MIC via the PCI-E bus is much slower than that of shared memory access [19]. Actually, CPU and MIC can be considered as different computing nodes rather than a shared memory node though they are indeed in a single node. So the pure multi-threaded ROSS is not compatible with the CPU+MIC collaborative computing platform. On the other hand, MPI based ROSS performs poorly on MIC, but MPI is a better way to support communication between CPU and MIC. This paper may be considered as the extension of our previous one. Taking advantages from both ROSS-MPI and ROSS-MT, we present a MPI+OpenMP version ROSS-OMPI, in which the collaborative work of CPU and MIC are implemented via MPI while the parallel processing is achieved via OpenMP multi-thread inside the MPI nodes.

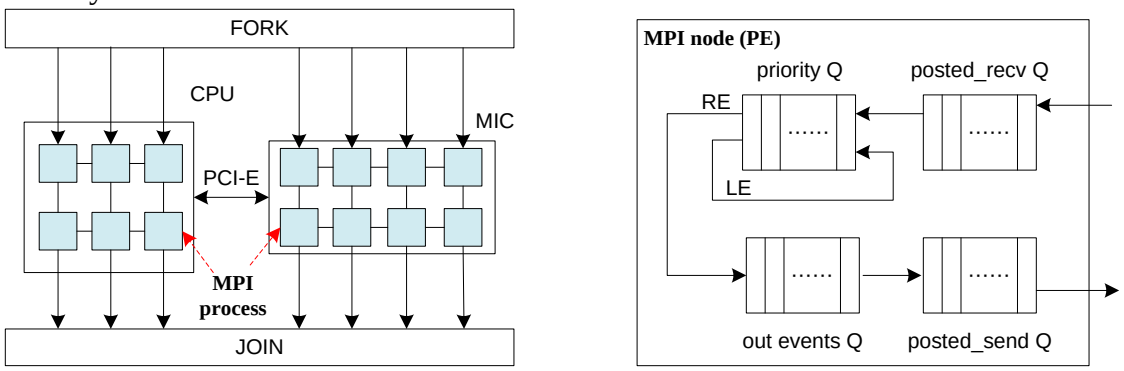
Background

MPI based ROSS on CPU+MIC In ROSS, processor (PE) is the execution unit to handle/send/receive events [18]. The MPI version of ROSS (allow us call it ROSS-MPI for short) uses an asynchronous communication mechanism to implement message sending and receiving among different MPI nodes. The MPI node is mainly the execution implementation for a single PE.

Fig.1 (a) shows the MPI fork-join process of ROSS-MPI on CPU+MIC. Obviously, there are several MPI nodes on both CPU and MIC respectively. Fig.1 (b) shows the events flow in ROSS-MPI briefly. Due to the asynchronous communication mechanism, the posted_recv queue is necessary for event receiving, same as the posted_send queue for event sending. For the event sending/receiving process may not complete. As a result, it may be too early to insert the received events into priority queue on receive end or free the events' memory on send end. The figure also shows that local events (LE) will be inserted into PE's priority queue directly.

As we presented in the previous paper, the multi-thread version ROSS performs pretty well on MIC while the MPI version performs very poor. However, MPI is a better way than MIC's offload mode when we consider the communication implementation between MIC and CPU. Before getting start, the performance of ROSS-MPI on multi-core CPU+MIC collaborating platform (see detail

information in Test results part) is tested. It is no surprise that the performance is even worse than that on MIC only. One of the reasons is the “cask effect”, while another is the additional delay introduced by communication between CPU and MIC.

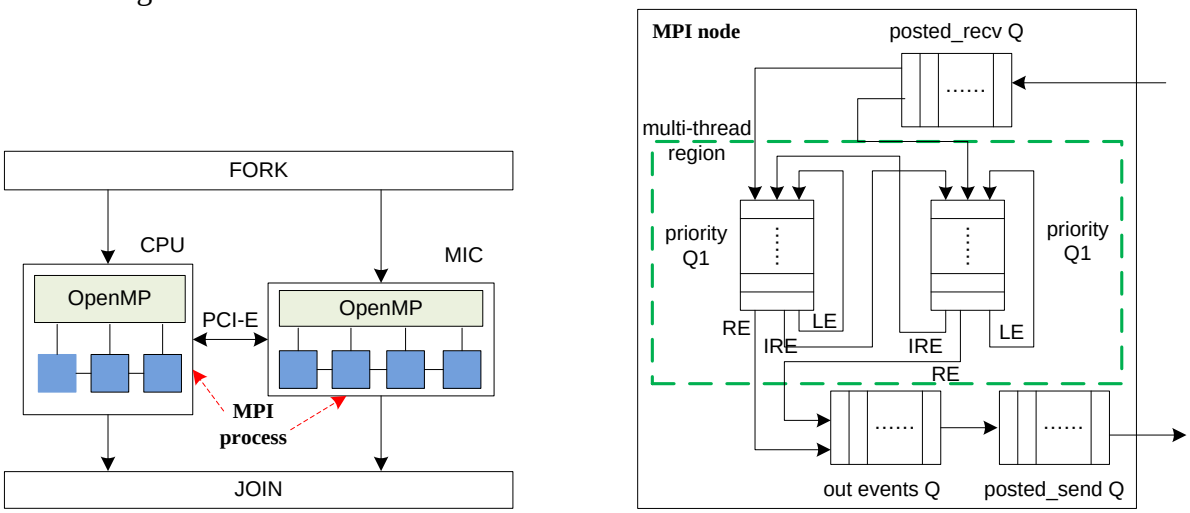


(a) ROSS-MPI on CPU+MIC (b) Events flow in ROSS-MPI

Fig.1. Schematic graph ROSS-MPI

MPI+OpenMP modeled ROSS on CPU+MIC

m. Based on ROSS-MPI, we introduced several simplified PEs in a MPI node, and issue a single OpenMP thread for each PE, which leads to a parallel processing of priority queues for a MPI node. Here, we call this MPI+OpenMP based ROSS as ROSS-OMPI for short. Fig. 2 (b) shows the schematic diagram of ROSS-OMPI on CPU+MIC.



(a) ROSS-OMPI on CPU+MIC (b) Events flow in ROSS-OMPI

Fig.2. Schematic graph of MPI+OpenMP modeled ROSS on CPU+MIC

The so called “simplified PE” means, compared with ROSS-MPI, PE in ROSS-OMPI only possesses a priority queue, while the out events queue, posted_recv queue and posted_send queue are shared resources which are mainly used to send and receive remote events from other node for all PEs in the MPI node. Consequently, there are three type events instead of two for ROSS-MPI. The in-node remote event (IRE.) presents events from a PE to another in the same MPI node. The IRE.s needs not to be send via MPI for all PEs’ priority queues are local resource, the source PE only needs to insert the IRE into the target PE’s priority. However, the target PE’s priority queue must be locked for other local PEs may send new events at the same time. The inter-node remote events (RE) are events from the other MPI nodes. The local event (LE) means the events for a PE itself. Fig. 2 (b) shows the brief progress of events handling/sending/receiving in ROSS-OMPI. The PEs are only response for sorting and handling events. For local events, the PE only needs to insert them into its priority queue directly. For remote events among MPI nodes, called the inter-node events, are sent and received by the MPI node. As a result, we can issue several OpenMP threads to handle the job of PEs, while the master thread will handle the sending and receiving of remote events.

Algorithm for main process loop of ROSS-OMPI

```
Input
   pe[]: array of PEs
   npe: number of PEs
   endtime: end time of simulation
Process
   1. Create npe OpenMP threads, each thread process pe[thread_id];
   2. while(true)
   3.    the master thread receives all the messages from the network via MPI, put them into each PE's priority
         queue;
   4.    the master thread compute the GVT;
   5.    if (GVT > endtime) break;
   6.    thread with id thread_id process events of pe[thread_id];
   7.    for each new event generated by pe[thread_id]
   8.       if the event is local, insert it into its own priority queue;
   9.       else if the event is inner-node remote event, insert it into target PE's priority queue;
  10.       else push it into out events queue
  11.    the master pull the events from out events queue and send them via MPI;
  12. end while
```

Fig.3. Algorithm for main process loop of ROSS-OMPI

In this version of ROSS-OMPI, we use a conservative time management strategy [17]. Fig. 3 shows the main process loop of ROSS-OMPI. However, for all PEs in a MPI node, receiving and sending of inter-node remote events and GVT (actually it should be called global LBTS, we just use GVT for convention) computation are still sequentially processed, which makes other threads waiting. This obstacle of exploiting parallelism leads to a bad performance of ROSS-OMPI, which is verified by experiment results in next section. However, there are a few optimization work can be conducted to make it better.

Test results

In this article, we use a computing node composed of two Intel Xeon E5 2650 multi-core CPUs (called the Host end) and one Intel Xeon Phi 3120 MIC coprocessor (called the MIC end). The MIC end is connected to the Host end via a PCI-E 2.0 interface. The schematic diagram of our hardware platform is illustrated in Fig. 4 and Table 1 shows the detailed configuration of our test platform, including both the hardware platform and software stacks.

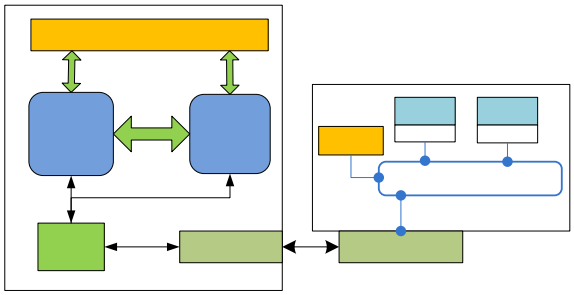


Fig.4. Schematic graph of CPU+MIC platform

Table 1 Detailed configuration of test platform

	Host	**Coprocessor**
Hardware Platform Configuration		
Processor Type	Intel Xeon E5-2650v2	Intel Xeon Phi 3100
Processors / cores	2 / 16	1 / 57
Memory Size	64 GB	6 GB
System Software Stack		
Operating System	CentOS 6.4	MPSS Update 3
C/C++ Compiler	Intel I_comp_xe 2013 sp1	
MPI Library	Intel MPI Library 5.0.0.028	

The PDES benchmark we use is PHOLD. PHOLD is one of the most classical benchmarks for PDES. In PHOLD, a set of Logical Processes (LPs) are distributed to different threads to process in parallel. At the beginning of the simulation, a number of initial events are scheduled by each LP directing to other LPs randomly. When a LP receives an event, it chooses another LP randomly and schedules a new event to the LP with random timestamp. So does the next LP. The simulation process will be terminated when the simulation end time comes. Generally speaking, the computation load of PHOLD model is quite light, so it is a communication dominant application. To some extent, many communication dominant PDES applications can be expressed by the PHOLD model. Furthermore, changing the relationship of LPs and adding additional computation load into

the model, performance evaluation and analysis with different purposes can be carried out.

Here, according to the results of our former article [16], we choose event granularity to be 10000 loops of Fused-Multiply-Add (FMA) operation, which is a relatively communication-computation balanced case.

Table 2 Parameters for PHOLD

Parameter	Value(s)
Number of LPs	3360
Start events/LP	200
Percent of remote event	0 0.25 0.5 0.75 1.0
Lookahead	1.0
End time	100.0
Event granularity	10000

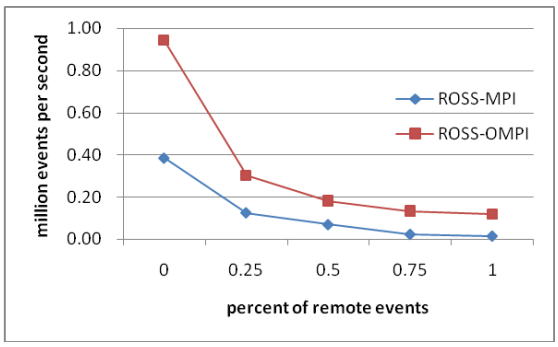


Fig.5. Experimental results for ROSS-MPI, ROSS-OMPI on CPU+MIC

In this experiment, we issue 16 and 56 OpenMP threads respectively on CPU and MIC. Fig. 10 shows the performance of ROSS-MPI, ROSS-OMPI and the optimized ROSS-OMPI on CPU+ MIC respectively.

Obviously, the optimized MPI+OpenMP model of ROSS does bring performance increase on CPU+MIC according to the results, by a factor of 2.4, 2.4, 2.6, 5.7 and 7.8 respectively.

Conclusion

The development of HPC platforms encourages us to do further researches on accelerating execution of complex PDES applications. Nowadays, the widespread of many-core processors brings new opportunities for us to enhance performance of PDES applications. MIC is a relative new many-core architecture compared with the widely-used GPU. In our previous paper, experiments for evaluating performance of ROSS are designed and tested on MIC and CPU respectively. However, no related works have been done on evaluation PDES simulators on CPU+MIC collaborative platform. Here we propose a PDES simulator based on MPI+OpenMP model and the PDES simulator ROSS which can be called as ROSS-OMPI. We use MPI to handle the interactive events between CPU and MIC, while several OpenMP threads are issued to process events in parallel inside CPU and MIC respectively. The experiment results show that ROSS-OMPI brings better performance.

Acknowledgement

We appreciate the support from National Natural Science Foundation of China (No. 61170048) and Research Project of State Key Laboratory of High Performance Computing of National University of Defense Technology (No. 201303-05).

References

[1] Yao Y, Zhang Y. Solution for Analytic Simulation Based on Parallel Processing [J]. Journal of System Simulation, 2008, 20(24): 6617-6621.

[2] Steinman J S. The WarpIV Simulation Kernel [C]. Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation, IEEE Computer Society, 2005, pp. 161-170.

[3] Ryan J M. Optimistic Parallel Discrete Event Simulation on a Beowulf Cluster of Multi-core Machines [D]. Cincinnati University, July, 2010.

[4] Jagtap D, Abu-Ghazaleh N, Ponomarev D. Optimization of Parallel Discrete Event Simulator for Multi-core Systems [C]. Proceedings of the 2012 ACM/IEEE/SCS 26th Workshop on Principles of Advanced and Distributed Simulation, IEEE Computer Society, 2012, pp. 520-531.

[5] Vitali R, Pellegrini A, Quaglia F. Towards Symmetric Multi-threaded Optimistic Simulation Kernels [C]. Proceedings of the 2012 ACM/IEEE/SCS 26th Workshop on Principles of Advanced and Distributed Simulation, IEEE Computer Society, 2012, pp. 211-220.

[6] Chen L, Lu Y, Yao Y, et al. A Well-Balanced Time Warp System on Multi-Core Environments [C]. Proceedings of the 2011 ACM/IEEE/SCS 25th Workshop on Principles of Advanced and Distributed Simulation, IEEE Computer Society, 2011, pp. 1-9.

[7] Perumalla K S. Discrete-event Execution Alternatives on General Purpose Graphical Processing Units (GPGPUs) [C]. Proceedings of the 20th Workshop on Principles of Advanced and Distributed Simulation, IEEE Computer Society, 2006, pp. 74-81.

[8] Park H, Fishwick P A. A GPU-Based Application Framework Supporting Fast Discrete-Event Simulation [J]. Simul-T Soc Mod Sim, 2010, 86(10): 613-628.

[9] Li X, Cai W, Turner S J. GPU accelerated three-stage execution model for event-parallel simulation [C]. Proceedings of the 2013 ACM SIGSIM conference on Principles of advanced discrete simulation, Montreal, Québec, Canada, 2013, pp. 57-66.

[10] Tang W, Yao Y. A GPU-Based Discrete Event Simulation Kernel [J]. SIMULATION: Transactions of the Society for Modeling and Simulation International.

[11] Jason L, Yuan L, Zhihui D and Ting L. GPU-Assisted Hybrid Network Traffic Model. Proceedings of the 2nd ACM/SIGSIM Conference on Principles of Advanced and Distributed Simulation ACM, 2014, pp. 63-74.

[12] Jagtap D., Bahulkar K., Ponomarev D., Abu-Ghazaleh N. Characterizing and Understanding PDES Behavior on Tilera Architecture [C]. Proceedings of the 2012 ACM/IEEE/SCS 26th Workshop on Principles of Advanced and Distributed Simulation IEEE Computer Society, 2012, pp. 53-62.

[13] Zhang Y, Wang J, Ponomarev D, Abu-Ghazaleh N. Exploring many-core architecture design space for parallel discrete event simulation [C]. Proceedings of the 2nd ACM SIGSIM/PADS conference on Principles of advanced discrete simulation ACM, 2014, pp. 95-104.

[14] Subhash S, Haoqiang J, Dennis J, Huiyu F, et al. An Early Performance Evaluation of Many Integrated Core Architecture Based SGI Rackable Computing System [C]. Proceedings of SC 13, 2013, pp. 1-13.

[15] Child R., Wilsey P. Dynamically Adjusting Core Frequencies to Accelerate Time Warp Simulations in Many-Core Processors [C]. Proceedings of the 2012 ACM/IEEE/SCS 26th Workshop on Principles of Advanced and Distributed Simulation IEEE Computer Society, 2012, pp. 35-43.

[16] Chen H, Yao Y, Tang W, Meng D, Zhu F and Fu Y. Can MIC Find its Place in the Field of PDES? An Early Performance Evaluation of PDES Simulator on Intel Many Integrated Cores [C]. Proceedings of 2015 IEEE/ACM 19th International Symposium on Distributed Simulation and Real Time Applications, 2015, pp. 13-16.

[17] Richard M F. Parallel and Distribution Simulation Systems. John Wiley & Sons, Inc., New York, NY, USA, 1999.

[18] Carothers C. D., Bauer D., Pearce S. ROSS: A high-performance, low-memory, modular Time Warp system [J]. Journal of Parallel and Distributed Computing, Academic Press, 2002(62): 1648-1669.

[19] Jim J, James R. Intel Xeon Phi Coprocessor High-performance Programming [M]. POST & TELECOM PRESS, Beijing, China, 2014.