A Study of the Effect of Partitioning on Parallel Simulation of Multicore Systems

Zhenjiang Dong, Jun Wang, George Riley, Sudhakar Yalamanchili
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332–0250
Email: zdong30, jun.wang, riley, sudha@ece.gatech.edu

***Abstract*—There has been little research that studies the effect of partitioning on parallel simulation of multicore systems. This paper presents our study of this important problem in the context of Null-message-based synchronization algorithm for parallel multicore simulation. This paper focuses on coarse grain parallel simulation where each core and its cache slices are modeled within a single logical process (LP) and different partitioning schemes are only applied to the interconnection network. In this paper we show that encapsulating the entire on-chip interconnection network into a single logical process is an impediment to scalable simulation. This baseline partitioning and two other schemes are investigated. Experiments are conducted on a subset of the PARSEC benchmarks with 16-, 32-, 64- and 128-core models. Results show that the partitioning scheme has a significant impact on simulation performance and parallel efficiency. Beyond a certain system scale, one scheme consistently outperforms the other two schemes, and the performance as well as efficiency gaps increases as the size of the model increases – with up to 4.1 times faster speed and 277% better efficiency for 128-core models. We explain the reasons for this behavior, which can be traced to the features of the Null-message-based synchronization algorithm. Because of this, we believe that, if a component has increasing number of inter-LP interactions with increasing system size, such components should be partitioned into several sub-components to achieve better performance.**

***Keywords*-parallel simulation; partitioning; multicore system; null message algorithm**

I. Introduction

Multicore designs for CPUs have been widely employed in recent years. Researchers and industry CPU architects use simulation to evaluate their design. So, simulation presently plays an important role in the field of multicore system design. The complexity of multicore system is high and sequential simulation for such systems can hit bottlenecks in terms of execution time. To handle the complexity and achieve scalability for simulation, [1] [2] [3] and [4] proposed parallel simulation for multicore system with parallel discrete even simulation (PDES) [5].

In PDES programs, simulation is separated into processes called logical process (LP). Each LP generates and processes events in a distributed manner by sending and receiving messages with time stamps and ensures global order of events. Synchronization algorithms play a key role and largely determine the performance of PDES programs. Various synchronization algorithms have been proposed by researchers, including the *Chandy-Misra-Bryant (CMB)* algorithm [6]. CMB is a conservative algorithm that guarantees the correct order of events. It uses Null messages to prevent deadlock. A Null message is a message that has no content. Its time stamp is the local time of the sender plus a positive value called lookahead and represents a lower bound for all future messages from the same sender. The purpose of Null messages is to help the receiver to advance its simulation time so that deadlocks are avoided. The CMB algorithm was implemented as one of the synchronization algorithms in the Manifold project [1].

Recently, [7] proposed an enhanced Null message algorithm for multicore system using domain specific knowledge and implemented it in the Manifold project. Experiments showed promising results for reducing the number of Null messages and speedup compared to serial simulation. However, [7] is mainly focused on the optimization of the Null message algorithm itself and doesn't explore the effect of partitioning on the different system models with such algorithms.

The purpose of this study is to examine the effect of partitioning on the performance and scalability of the Null-message-based parallel simulation of multicore systems. In the following sections we will first show the effect that three different partitioning schemes have on 16-, 32-, 64- and 128-core system models. Then we will analyze the reason for such effects based on the features of Null-message-based synchronization algorithm. Finally, following this reason we will present our conclusions as a general guide for partitioning of Null-message-based parallel simulation.

II. Background and Related Work

The Manifold project is an open source software project that provides a scalable infrastructure for modeling and simulation of many-core architectures [1]. It uses PDES, and a component based design and standardized interfaces. A typical system model that Manifold is designed to simulate is presented in Figure 1. As shown in Figure 1, the system model consists of a certain number of cores, caches, memory controllers, and one interconnection network. Each core is connected to a cache, and then cache and memory controllers connected to an interconnection network. The dashed line shows one partitioning scheme we will cover in this paper.

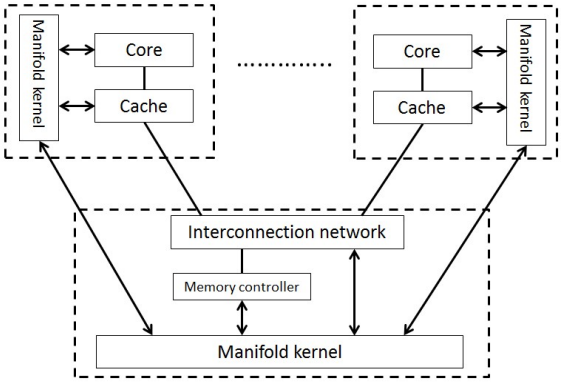

Figure 1: Typical system model of Manifold with the baseline partitioning.

The kernels interact with each other and keep synchronization for all components within its LP. The Manifold project provides two major categories of synchronization algorithms, which include a barrier based algorithm called LBTS [5] and the CMB algorithm. The following discussion is based on the enhanced CMB algorithm proposed in [7] e.g. *Forecast Null Message algorithm (FNM)*.

There has been extensive research on the problem of partitioning general networks, particularly VLSI systems. For example, Karypis and colleagues developed the hMETIS [8] partitioning package for VLSI systems, and hMETIS achieved 9%-30% [9] performance gain compared to other schemes they evaluated in [9]. And [9] confirmed partitioning can have significant impact on performance of parallel simulation for large scale systems. However their work focuses on parallel simulation of large scale systems such as VLSI that has components in the order of hundreds of thousands, which is different from multicore systems with much smaller scale that we discuss here.

III. Partitioning Schemes and Design of Experiments

Currently Manifold does not use any automatic partitioning tool, mainly because the scale of the systems Manifold is designed to simulate is relatively small compared to systems like VLSI. In this study, we compare three different manual partitioning schemes, examine their effect on simulation performance and parallel efficiency, and explain the reason for the effect.

A. Partitioning Schemes

The multicore systems we consider have a star topology consisting of an interconnection network and a number of nodes connected to the network. Each node contains either a memory controller or a core and its cache slices. The inter-component links that can be cut by a partitioning scheme fall into the following categories:

- Core-cache links.
- Cache-network links.
- Memory controller-network links.
- Router-router links.

We do not cut the core-cache links, due to the fact that cache hit rate is generally well above 95%. In the experiments we conducted, the average number of messages on the cache-network links is less than 1.7% of that of the core-cache links. Therefore we always cut the cache-network links but do not cut the core-cache links. We do not discuss the effect of assigning multiple cores and caches to a single LP and leave this problem as future work. On the other hand, the memory controller models that we use are rather simple, so each memory controller is assigned to the same LP as the router to which it is connected.

According to above discussion, it leaves only the question of whether the router-router links should be cut. The interconnection network in a multicore system is responsible for communication among caches to maintain cache coherence and it also transfers memory accesses between memory and caches. Intuitively, the network should be divided into multiple parts. Due to the features of Null-message-based simulation, an LP's ability to make progress depends heavily on the messages it receives from all its input channels. The more input channels it has, the longer it has to wait for all the input channels to have new messages, and the more messages it needs to process. Therefore, in general, having more input channels is detrimental to the performance of a Null-message-based parallel simulation. The three partitioning schemes we study in this paper differ only in how the network is partitioned.

1) Scheme 1: 1-part: The first partitioning scheme, which also is the baseline scheme, can be seen in Figure 1. It is the partitioning scheme adopted in [7]. In this scheme, the entire interconnection network and all memory controllers are assigned to a single LP. For simulations utilizing this partitioning scheme, *number of cores + 1* LPs are required, e.g. 17 LPs for 16-core system and so on.

2) Scheme 2: 2-part: In this partitioning scheme we divide the interconnection network into two equal halves, and each half is assigned to one LP. This scheme can be seen in Figure 2. It is straightforward to see *number of cores + 2* LPs are required.

3) Scheme 3: Y-part: As shown in Figure 3, the *Y-part* scheme separates the network into Y parts, where Y equals the value of the *y* dimension in an $X \times Y$ torus network. Each row of routers is assigned to one LP, and each network LP is connected to roughly X core-cache LPs. In this partitioning scheme, *number of cores + Y* LPs are required.

B. Design of Experiments

The system models we built and tested included multicore systems with 16, 32, 64 and 128 cores. For each system model, all cores and their caches are connected to a single interconnection network, and one memory controller is

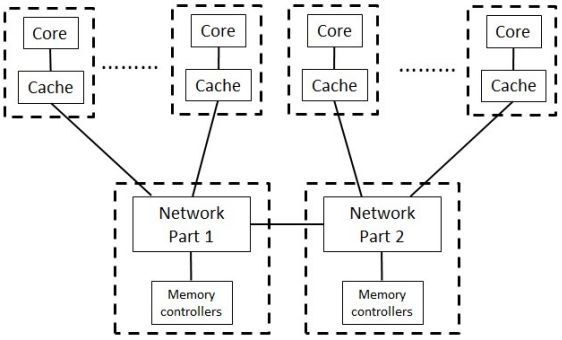

Figure 2: 2-part scheme.

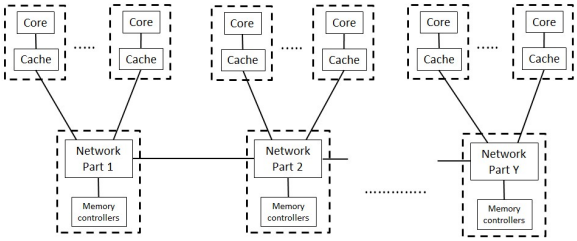

Figure 3: Y-part scheme.

created for every 8 cores in the systems. The underlying interconnection networks we use in our systems are torus networks. And, 5×4, 6×6, 9×8 and 12×12 torus networks are built for 16-, 32-, 64- and 128-core systems respectively. The core model we use in this paper is a cycle-accurate out-of-order x86 model called Zesto [10]. Each core component has a private write-back L1 cache and a shared write-back L2 cache slice [11]. Cashes implement the Modified-Exclusive-Share-Invalid (MESI) coherence protocol [12]. All the architecture components in our system are registered to the same clock and run at the same frequency. A credit based flow control protocol has been employed along the core-cache-network path and among routers inside the network. Simulations are conducted on a Linux cluster that has two Intel Xeon X5670 6-core CPUs with 24 hardware threads on each node. The operating system is RHEL release 6.3, and MPI version is Open MPI 1.5.4. In all tests, we use enough hardware resources such that each LP has its own hardware thread. For each combination of system model and partitioning scheme, tests have been conducted with 5 PARSEC benchmarks [13], which include *facesim*, *ferret*, *freqmine*, *streamcluster* and *vips*.

In our tests, total time consumption *(t_total)* and time consumed by each LP to gather and process Null messages *(t_gp)* are recorded for each run of test. We utilize Linux *time* command to record *t_total* and *clock_gettime()* [14] function to get the *t_gp*. In the function that gathers and processes Null messages, we insert two calls of the *clock_gettime()* function, one at the beginning and one at the end, to measure the execution time of the function. The execution time is accumulated in *t_gp*.

IV. Experimental Results

The experiment results show clearly the effect of partitioning, and also indicate the *t_gp* is the main factor that determines the performance beyond certain system size.

A. Time Consumption and Comparison of Schemes

Tables I, II, III and IV show the total time consumption to run simulation and the speedup compared to serial simulation (inside the parentheses) for each system model and benchmark. From these 4 tables we can see the speedup keeps growing with increasing system scale for *Y-part*, however, for the other two schemes, the speedup goes down when system scale reaches 64 cores.

As shown in Table I, in tests of the 16-core system the difference in performance of the partitioning schemes is relatively small. *Y-part* always has best performance that is 1% to 9% better than the others, *1-part* and *2-part* have very similar performances with a difference of about 1%.

Table II shows experiment results for tests of 32-core system. From this table we can see *Y-part* consumes at least 14% less time than *1-part* and 7% less time than *2-part*. Different from the tests for 16-core system, in tests for 32-core system *2-part* consistently outperforms *1-part* by 3% to 9%.

The performance gap between partitioning schemes grows enormously in tests for 64-core system. From Table III, we can observe that *Y-part* achieved about 2 times faster speed than *1-part* in 4 out of 5 benchmarks, and 1.7X faster speed for *vips*. Additionally, it also outperforms *2-part* by 30% to 67%. At the same time, *2-part* runs more than 30% faster than *1-part*.

From Table IV we can see, the performance gap is further increasing in tests for 128-core system. *Y-part* runs at least 3.5 times faster than *1-part*, and compared to *2-part* it achieves 2.2 to 2.9 times faster speed. Very similar to tests for 64-core system, *2-part* has 26% to 42% less time consumption than *1-part*.

Table I: Simulation running time in seconds for 16-core system.

Benchmarks	Seq.	1-part	2-part	Y-part
facesim	3996	735 (5.4X)	734 (5.4X)	696 (5.7X)
ferret	4021	818 (4.9X)	820 (4.9X)	750 (5.4X)
freqmine	4155	756 (5.5X)	750 (5.5X)	707 (5.9X)
streamcluster	4089	739 (5.5X)	739 (5.5X)	718 (5.7X)
vips	4116	720 (5.7X)	730 (5.6X)	708 (5.8X)

We compare the parallel efficiency of the partitioning schemes in a normalized manner using the following equation:

$$\textit{Normalized Efficiency} = \frac{\frac{t\_total\_1-part}{t\_total}}{\frac{num\_lp}{num\_LP\_1-part}} \quad (1)$$

Table II: Simulation running time in seconds for 32-core system.

Benchmarks	Seq.	1-part	2-part	Y-part
facesim	9023	1241 (7.3X)	1192 (7.6X)	1073 (8.4X)
ferret	8600	1531 (5.6X)	1368 (6.3X)	1087 (7.9X)
freqmine	8555	1241 (6.9X)	1208 (7.1X)	1085 (7.9X)
streamcluster	8748	1210 (7.2X)	1180 (7.4X)	1044 (8.4X)
vips	8634	1230 (7.0X)	1121 (7.7X)	1046 (8.2X)

Table III: Simulation running time in seconds for 64-core system.

Benchmarks	Seq.	1-part	2-part	Y-part
facesim	18163	3740 (4.9X)	2656 (6.8X)	1592 (11.4X)
ferret	18561	3632 (5.1X)	2662 (7.0X)	1716 (10.8X)
freqmine	17959	3753 (4.8X)	2753 (6.5X)	1956 (9.2X)
streamcluster	14079	3786 (3.7X)	2796 (5.0X)	1813 (7.8X)
vips	18000	3619 (5.0X)	2717 (6.6X)	2080 (8.7X)

where *t_total_1-part* and *num_LP_1-part* are the total time consumption and number of LPs used in tests of *1-part*, and *t_total* and *num_LP* are the total time consumption and number of LPs used in the partitioning scheme in question. It's easy to see with equation (1) the efficiency of *1-part* is normalized to 1. As, shown in Figure 4, *1-part* has the best efficiency for 16-core system, as it achieved similar performance with fewer LPs. However, *2-part* and *Y-part* outperform *1-part* in tests for system models with larger scale. Based on these results, we can see good partitioning scheme also improves efficiency greatly when system reaches a certain scale.

B. Relation between Total Time Consumption and Time Consumed by Network LP(s) to Process Null Messages

From Table I to Table VIII we can observe that *t_gp* of *1-part* and *2-part* increases much faster than the total time consumption, and this results in increasing of percentage of *t_gp* out of the total running time. In tests using *1-part* the average percentage of *t_gp* increase sharply from 18% to 49% when system scale grows from 16-core to 128-core. Similarly, for *2-part*, when system scale increases from 16 to 128 cores, percentage of *t_gp* increases from 17.7% to 37.4%. On the other hand, *Y-part*'s *t_gp* and total time consumption increase roughly at the same speed as the system scale. Its percentage of *t_gp* stays below 20% in tests for all system models. The results indicates that if we do not partition the network LP, the number of inter-LP links that the network has grows with growing system scale, making the amount of Null messages that the network LP needs to handle also increase. Therefore, the overhead incurred on the network LP to gather and process Null messages increases greatly. Eventually, the *t_gp* becomes the major impediment for performance.

Following the discussion above, we believe that in Null-message-based parallel simulation of multicore system, when the system scale reaches a certain level, partitioning the interconnection network is critical to achieving good performance. Further, based on our study, we can draw the conclusion that, in a Null-message-based parallel simulation, if there are components whose number of inter-LP links increases as system scales, then such components should be partitioned into multiple parts to reduce the overhead of Null messages and to achieve better scalability.

Table IV: Simulation running time in seconds for 128-core system.

Benchmarks	Seq.	1-part	2-part	Y-part
facesim	30295	7258 (4.2X)	4570 (6.6X)	2029 (14.9X)
ferret	30410	8455 (3.6X)	4864 (6.2X)	2062 (14.7X)
freqmine	29604	7517 (3.9X)	5551 (5.3X)	1936 (15.9X)
streamcluster	36300	7281 (5.0X)	5335 (6.8X)	2054 (17.7X)
vips	32413	7396 (4.4X)	4858 (6.7X)	2076 (15.6X)

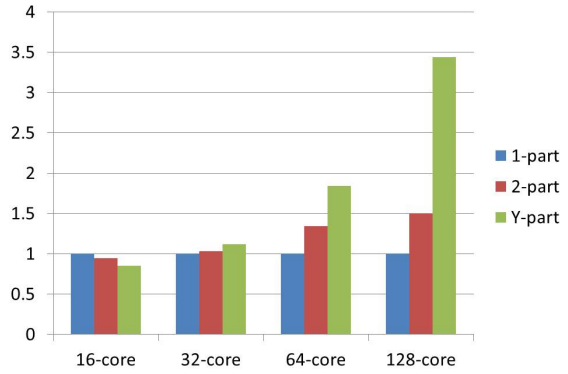

Figure 4: Normalized efficiency of partitioning schemes.

Table V: *t_gp* and its percentage out of total time for 16-core system.

Benchmarks	1-p.	Per. 1-p.	2-p.	Per. 2-p.	Y-p.	Per. Y-p.
facesim	131	17.8%	133	18.1%	132	19.0%
ferret	185	22.6%	151	18.5%	136	18.1%
freqmine	137	18.0%	146	19.4%	130	18.4%
streamcluster	123	16.6%	122	16.6%	113	15.8%
vips	118	16.4%	117	16.0%	115	16.3%

Table VI: *t_gp* and its percentage out of total time for 32-core system.

Benchmarks	1-p.	Per. 1-p.	2-p.	Per. 2-p.	Y-p.	Per. Y-p.
facesim	265	21.3%	270	22.7%	165	15.4%
ferret	426	27.8%	323	23.6%	165	15.2%
freqmine	271	21.9%	284	23.5%	127	11.7%
streamcluster	213	17.6%	258	21.9%	123	11.8%
vips	232	19.0%	232	20.7%	118	11.3%

V. Conclusion and Future Work

This paper presents our study of partitioning multicore systems for parallel simulation. Three partitioning schemes are investigated. Our study shows partitioning schemes

Table VII: *t_gp* and its percentage out of total time for 64-core system.

Benchmarks	1-p.	Per. 1-p.	2-p.	Per. 2-p.	Y-p.	Per. Y-p.
facesim	1635	43.7%	822	30.9%	333	20.9%
ferret	1528	42.0%	845	31.7%	325	18.9%
freqmine	1684	44.9%	904	32.8%	315	16.1%
streamcluster	1651	43.6%	914	32.7%	328	18.1%
vips	1565	43.3%	833	30.7%	294	14.1%

Table VIII: *t_gp* and its percentage out of total time for 128-core system.

Benchmarks	1-p.	Per. 1-p.	2-p.	Per. 2-p.	Y-p.	Per. Y-p.
facesim	3365	46.4%	1912	41.8%	429	21.1%
ferret	4096	48.4%	1990	40.9%	446	21.6%
freqmine	3751	50.0%	1966	35.4%	385	19.9%
streamcluster	3567	49.0%	1966	36.8%	399	19.4%
vips	3558	48.1%	1876	38.6%	425	20.5%

can have significant impact on the performance of Null-message-based parallel simulation of multicore system. And, the effect of partitioning becomes increasingly important as the scale of the simulated system grows. In the multicore systems we study, the interconnection network is connected to a number of core-cache nodes. As system size grows, this number also increases. If the network is not partitioned the overhead incurred on the network LP to process Null messages grows exponentially, leading to great performance degradation. Partitioning the network can reduce this overhead and helps performance. Of the three partitioning schemes we study, *2-part* is only one step in the right direction. The *Y-part* scheme, which partitions an $X \times Y$ torus network into Y parts, appears to be a scalable solution, and outperforms the other two not only in simulation speed but also in parallel efficiency when the system scale reaches a certain level. Based on this study, we believe, in a Null-message-based parallel simulation, to achieve reasonable scalability, any component whose inter-LP links increases with system size should in principle be partitioned.

For our future work, we intend to continue our study of scalability of parallel multicore simulation and apply what we have learned to even bigger models, i.e., models with 256 cores and beyond. Another important problem is to determine the optimal number of cores and caches to assign to a single LP, and explore the optimal value for Y in the Y-part scheme.

REFERENCES

[1] manifold.gatech.edu, "Manifold," http://manifold.gatech.edu.

[2] A. Rodrigues, K. Hemmert, B. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Risen, J. Cook, P. Rosenfeld, E. CooperBalls, and B.Jacob, "The structural simulation toolkit," *ACM SIGMETRICS Performance Evaluation Review*, vol. 38, no. 4, pp. 37–42, March 2011.

[3] J. Miller, H. Kasture, G. Kurian, C. Gruenwald, N. Beckmann, C. Celio, J. Eastep, and A. Agarwal, "Graphite: A distributed parallel simulator for multicores," *Proceedings of the 16th International Symposium on High-Performance Computer Architecture*, pp. 1–12, 2010.

[4] J. Chen, L. Dabbiru, D. Wong, M. Annavaram, and M. Dubois, "Adaptive and speculative slack simulations of cmps on cmps," *Proceedings of the 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 523–534, 2010.

[5] R. Fujimoto, *Parallel and Distributed Simulation Systems*. John Wiley & Sons, 2000.

[6] K. Chandy and J. Misra, "Distributed simulation: a case study in design and verification of distributed programs," *IEEE Transactions on Software Engineering*, vol. SE-5, no. 5, pp. 440–452, 1979.

[7] J. Wang, Z. Dong, S. Yalamanchili, and G. Riley, "Optimizing parallel simulation of multicore systems using domain-specific knowledge," *2013 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (PADS)*, pp. 127–136, 2013.

[8] http://glaros.dtc.umn.edu, "hmetis - hypergraph and circuit partitioning," http://glaros.dtc.umn.edu/gkhome/metis/hmetis/overview.

[9] G. Karypis, R. Aggarwal, and V. Kummar, "Multilevel hypergraph partitioning: Applications in vlsi domain," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 1, 1999.

[10] G. Loh, S. Subramaniam, and Y. Xie, "Zesto: A cycle-level simulator for highly detailed microarchitecture exploration," *International Symposium on Performance Analysis of Software and Systems*, pp. 53–64, 2009.

[11] J. Wang, J. Beu, S. Yalamanchili, and T. Conte, "Designing configurable, modifiable and reusable components for simulation of multicore systems," *3rd International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS12)*, pp. 472–476, November 2012.

[12] M. S. Papamarcos and J. H. Patel, "A low-overhead coherence solution for multiprocessors with private cache memories," *ISCA'98 25 years of the international symposia on Computer architecture*, pp. 284–290, 1998.

[13] C. Bienia and K. Li, "Parsec 2.0: A new benchmark suite for chip-multiprocessors," *Proceedings of the 5th Annual Workshop on Modeling, Benchmarking and Simulation*, 2009.

[14] http://www.qnx.com, "clock_gettime," http://www.qnx.com/developers/docs/6.4.1/neutrino/lib_ref/c/clock_gettime.html.