# CRAW/P: A Workload Partition Method for the Efficient Parallel Simulation of Manycores

Shuai Jiao[1,2], Paolo Ienne[3], Xiaochun Ye[1], Da Wang[1], Dongrui Fan[1], and Ninghui Sun[1]

[1] SKL Computer Architecture, ICT, CAS, Beijing, P.R. China
[2] Graduate University of Chinese Academy of Sciences, Beijing, P.R. China
[3] École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland
{jiaoshuai,yexiaochun,wangda,fandr,snh}@ict.ac.cn, Paolo.Ienne@epfl.ch

**Abstract.** This paper addresses the workload partition strategies in the simulation of manycore architectures. The key observation behind this paper is that, compared to traditional multicores, manycores feature more non-uniform memory access and unpredictable network traffic; these features degrades simulation speed and accuracy of *Parallel Discrete Event Simulators (PDES)* when one uses static workload partition schemes. Based on the observation, we propose an adaptive workload partition method: *Core/Router-Adaptive Workload Partition (CRAW/P).* The method delivers more speedup and accuracy than static partition schemes by partitioning the simulation of on-chip-network independently from that of the cores and by synchronizing them differently. Using a PDES simulator, we evaluate the performance of CRAW/P in simulating a 256-core general purpose many-core processor. Running SPLASH2 benchmark applications, the experimental results demonstrate it can deliver speed improvement by 28%~67% over static partition scheme and reduces timing errors to <10% in very relaxed simulation (quantum size as 64).

**Keywords:** Parallel Simulation, Manycore, Multicore, Workload Partition.

## 1 Introduction

While the "manycore era" approaches, some manycore processors have already arrived [1, 2, 3, 4, 5]. Thousand-core processors are no longer infeasible, and it is likely that thousands of cores on a single die will become a commodity [6]. Simulating such parallel systems is a serious problem. Currently, the majority of simulators available are sequential [7, 8, 9] and thus run the simulation workload on a single host thread. When the number of cores increases in a target system, the simulation performance for each core goes down.

A variety of techniques have been proposed to accelerate simulation. These techniques include the following: parallel simulation [10, 11, 12, 13, 14, 15], direct execution [15, 16], and FPGA acceleration [17, 18, 19]. Among these techniques, parallel simulation speeds simulation by exploiting the parallelism inherent in the target parallel architecture. Besides, the advent of low-cost SMP computers makes

© Springer-Verlag Berlin Heidelberg 2012

parallel simulation very attractive. Representative works on parallel simulation are *Parallel Discrete Event Simulation (PDES)* simulators [21].

State-of-the-art PDES simulators focus on simulating multicores. These simulators partition the simulation workload in a simple static manner. Examples could be found in P-mambo [16], SlackSim [14], and Graphite [15]. In these simulators, target cores are evenly distributed among host threads. These schemes work well in simulating multicores but are inefficient in simulating manycores. The distinction lies in the architectural difference between multicores and manycores: manycores feature large scale on-chip-networks, which produce more non-uniform memory accessses (NUCA) and unpredictable network traffic.

Based on these observations, this paper proposes a partition method: core/router-adaptive workload partition (CRAW/P). The essential idea of CRAW/P is that it partitions the simulation workload adaptively, divides the simulation of on-chip-network (routers) separately from that of the cores and simulates network more strictly. The method delivers more speedup and accuracy than static partition schemes.

The main contributions of CRAW/P are 1) how to use adaptive partitions to deliver speedup and accuracy and 2) how to leverage the core/router partitioning to efficiently maintain accuracy in terms of (i) reduction of host threads that simulate the network and (ii) strict synchronization for the network. As far as we can tell, we are the first to comprehensively discuss the workload partition scheme in manycore simulation and exploit simulation speedup and accuracy saving from the division of network and cores in parallel simulating manycores.

The remainder of this paper is structured as follows. Section 2 discusses the architectural characteristics of manycores and demonstrates the observations. Section 3 discusses the details of CRAW/P. Section 4 illustrates the experimental results. Section 5 discusses some related work. Section 6 offers some concluding remarks.

## 2 Observation

This section explains why static workload partition scheme is far from an optimal choice for simulating manycores and demonstrates the observation that motivates the adaptive partitioning and the core/router partitioning. Some experimental examples are given using the experimental platform later described in Section 4.

Fig. 1 shows how static partition schemes are used to simulate typical multicores and manycores. The partition scheme in Fig. 1(a) represents the scheme used, for instance, by SlackSim [14] while the partition scheme in Fig. 1(b) represents what used in Graphite [15]. The static scheme may work well in these multicore simulators but it is inefficient for manycores. The key is the large-scale on-chip-network, which is widely used in modern manycore architectures to provide better scalability. A large on-chip-network enlarges the following phenomenon on manycores: non-uniform cache accesses and unpredictable network traffic.

***Non-uniform cache access (NUCA)*** produces an obvious workload imbalance. On large scale networks, NUCA accesses might produce very non-uniform on-chip traffic due to topology and routing. The situation is worsened if hot spotting appears on the network. The non-uniform traffic results in some cores being busy running ahead due to short off-core latency, while other cores are stalling for reply. In a typical PDES simulator, simulating a busy core/router results in a large workload while simulating a

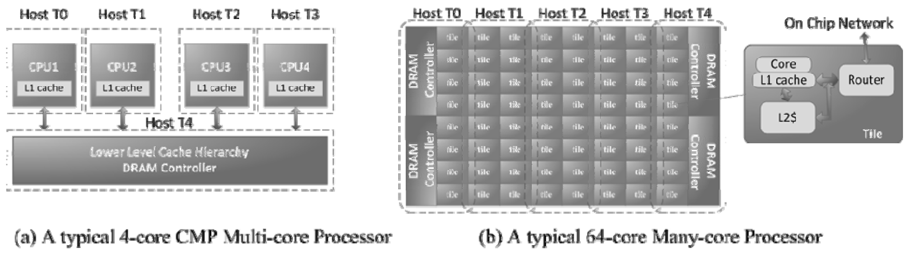


**Fig. 1.** A typical multicore SMP and manycore processor simulated by 5 host threads using static partition schemes. The Multi-core architecture is typical CMP architecture. Each core of a CMP has a L1 instruction and coherent data cache. The lower level cache hierarchy is made of L2 (or even L3) cache banks, which are accessed in a NUCA [21] manner. Banks can be private or shared to each core. The manycore architecture is extracted from state-of-the-art manycore processors (e.g., Intel SCC [1], Polaris [2], Tile64/Pro/Gtx [3, 4], and Godson-T [5]). These processors present common features such as distributed caches, mesh network, and DRAM controllers at the peripheral of the chip.

waiting core/router involves practically no activity. Fig. 2 shows the workload distribution when simulating a 256-core architecture running the *matrix_multiply* kernel. The arrows and circles indicate that on-chip cache-access hot spotting is the main reason of non-uniform workload distribution.

If the simulation workload is statically partitioned among host threads, the workload imbalance would slowdown simulation speed because host threads spend more time on synchronizing their local clocks. Besides, in cases of relaxed simulation (e.g., quantum method), clock skew is more likely to happen, resulting in more timing errors. Fig. 3 shows a example of static partition scheme which demonstrates the relation of simulation speed and timing error with workload imbalance. *MCPS (Million Cycles per Second)* is the measure of simulation speed: it indicates how many target cycles the simulator achieves during one wall-clock second. *ECPE (Error Cycles per Event)* is the measure of the dynamic timing error during simulation run time: it indicates the average cycle deviation of an event timestamp from the cycle it actually takes effect. Workload imbalance is measured by the *Standard Deviation (SD)* of the synchronization overhead for all host threads. As illustrated, simulation speed presents opposite variation against workload imbalance while timing error varies similarly with the imbalance. The relation is more obvious in the region of 130–150s.

**Unpredictable network traffic** makes parallel simulation of network quite sensitive to accuracy. The network connects so many interacting tiles that we cannot predict the on-chip traffic pattern. Besides, most modern on-chip routers achieve one-cycle latency between neighboring tiles; as a result, there is no "critical-latency" (SlackSim) for any router-to-router link. In this situation, if neighboring routers are asynchronously simulated by different threads, even a limited clock skew between host threads would easily produce timing errors.

Further examining the network traffic, we find that the traffic between core and router is less frequent and more deterministic than that between router and router. A *Router-Router (RR)* link would be quite busy even if all cores inject infrequently messages into the network because a router-router link may be shared by many interacting cores. The *Core-Router (CR)* link, however, demonstrates some

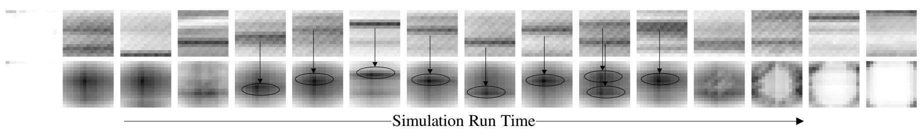


**Fig. 2.** Relation between on-chip hot spotting in accessing shared cache (top) and workload distribution (bottom) during simulation. Data is collected from the simulation of a 256-tile manycore processor running *matrix-multiply*. Hot spotting is measured as the busy cycles of every L2$. Workload is measured by the time spent in simulating each tile. Each pane (16x16 mini blocks) represents a snapshot of simulation workload distribution. Each gray mini block in the pane indicates a target tile. The shade of gray indicates the workload level: darker is for heavy workload while lighter for light workload.

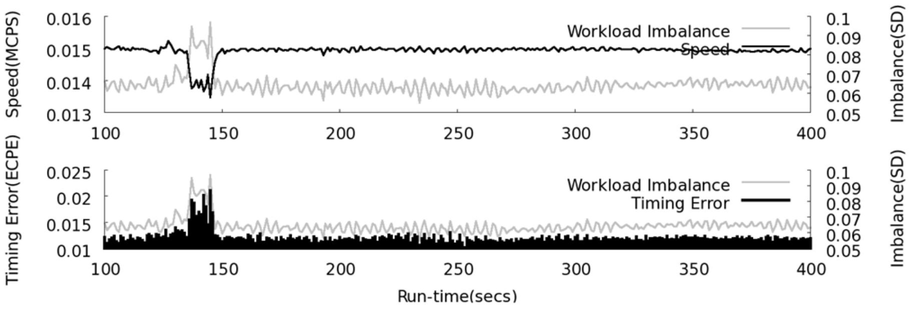
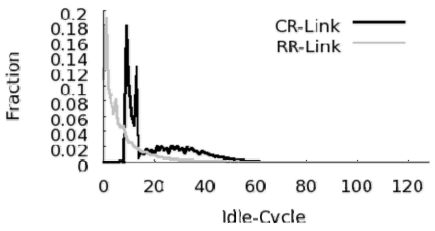


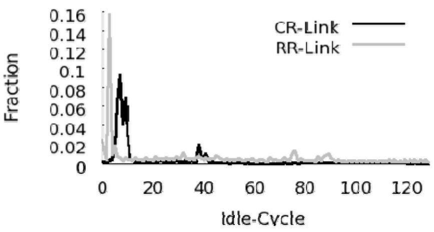
**Fig. 3.** Simulation speed (MCPS) and timing error (ECPE) are influenced by workload imbalance (Standard Deviation) during simulation run-time. Data is collected from a simulation section of *matrix-multiply* by 8 host threads with a relaxed quantum of 16. An obvious episode of workload imbalance is observed in the region of 130–150s.

determinism. For example, we can predict the most probable interval between two continuous Cache-Misses on a CR link. Also, we can determine the minimum interval between a Cache-Miss event and the corresponding Cache-Refill event on a CR link because the L2$ (Level 2 cache) has the minimum processing cycles for a Cache-Miss request. Although these events may happen concurrently and result in less cycles between two CR link events, the CR link still present more determinism than RR link.

To demonstrate the difference between RR link and CR link, the idle-cycle distribution of RR link and CR link is presented in fig. 4. An idle-cycle is defined as the cycles between two continuous traffic events on a specific link. For example, an idle-cycle of 4 for a CR link indicates that a link event happens 4 cycles after the previous one on that CR link. Two simulation cases are demonstrated in Fig. 4: *matrix_multiply* and *lu*. *Matrix_multiply* is a highly parallel application and contains little inter-core communication. Most network traffic events are Cache-Misses/Refills. *Lu* is among the applications of the SPLASH2 benchmark which contain most inter-core communication. Most of the on-chip traffic is coherence traffic between caches. Both simulations show that router-router link is busier than core-router link. For example, in *matrix_multiply*, the idle-cycles equal to 2 accounts for a proportion of



(a) *matrix_multiply*



(b) *lu*

**Fig. 4.** Idle-cycles distribution in the simulation of a 256-core architecture running *matrix_multiply* and *lu*. Idle-cycles larger than 128 is ignored.

20%. The idle-cycles for the CR link, however, mostly falls beyond 8, which is determined by the least processing cycles of Cache-Miss message in L2$ (L2$ hit latency). In *lu*, idle-cycles mainly fall in the region of 4–10, which is determined by the minimum processing cycles of Cache-Invalidation message in the core.

## 3 CRAW/P

This section discusses the proposed partition scheme, CRAW/P. Two partition schemes are also described for comparison: Static and Simple-Adaptive. Fig. 5 shows examples of these three partition schemes. In these schemes, simulation of the target is distributed onto multiple host threads. Host threads run in a relaxed PDES manner. Implementation details of the used PDES method in this paper could be referred in [22].

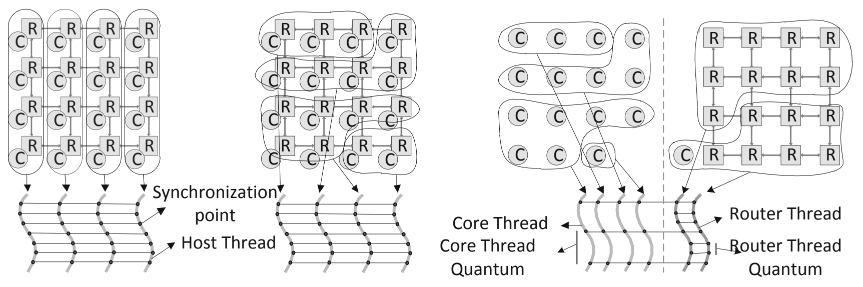


**Fig. 5.** Workload partitioning for Static, Simple-Adaptive, and CRAWP. Note that Static and Simple-Adaptive use one synchronization quantum among all host threads while CRAWP uses two different quantum sizes: core thread quantum (large) and router thread quantum (small).

The described Static partition scheme is exactly the scheme used in Graphite. It simply partitions the target tiles evenly onto all host threads. The Simple-Adaptive scheme is a very straightforward manner to create adaptive partitions: It tries to maintain workload balance between all host threads. As a result, host threads spend less time on synchronization, resulting in some speedup. Besides, better balance produces less clock-skew between host threads, reducing timing errors in a relaxed simulation.

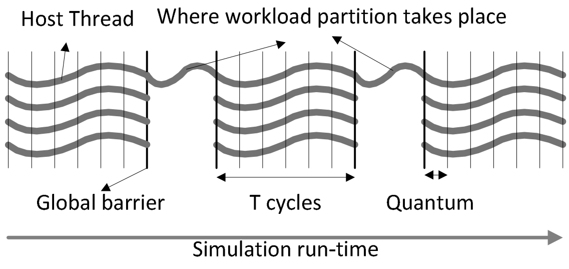


**Fig. 6.** Adaptive workload partition in Simple-adaptive and CRAW/P

Fig. 6 shows how adaptive partition works in a quantum based PDES simulator. During the simulation, host threads record the simulation time for each tile; all host threads barrier twice after a period of T cycles; between every barrier-pair, only one host thread runs the repartition work while other threads are simply waiting. The partitioning thread assembles the summary workload and redistributes the tiles to each thread, assuring workload balance. The idea of adaptive partition is to use the workload distribution in the near history to guide the workload partition in the near future and achieve better balance. In practical simulation, the partition interval should be set to a proper value: a big interval would result in bad partition efficiency while a small one would introduce considerable synchronization overhead. In this paper, the interval is set dynamically during simulation to ensure the partition overhead is less than a constant (e.g. 2%).

CRAW/P is essentially an adaptive partition scheme and has the same basic mechanism as Simple-Adaptive. However, it goes further; it partitions the network apart from the cores and allows the simulation of network and cores to use different synchronization strategies. To implement the network/core partition and different synchronization strategies, CRAW/P divides the host threads into two types: core thread and router thread. Core threads simulate cores (processing pipeline, L1$, and L2$) and synchronize with a coarse quantum; on the other hand, router threads simulate routers and synchronize with a fine quantum. The idea of CRAW/P could be defined through the following constraints on the simulator:

1. **Workload balance must be maintained between all host threads (core threads + router threads).** Since workload balancing is the basic mechanism to reduce synchronization overhead and clock-skew, it is a must for parallel simulation.

2. **Network must be simulated by router threads.** Since the network is quite sensitive to parallelism, the network should be simulated by as few threads as possible. Because the router is a light weight module compared to core module (functional model and core timing model), this constraint will largely reduce the number of router threads.

3. **Cores should be simulated by core threads with higher priority than by router threads.** Ideally, cores are simulated only by core threads. However, in cases of serious workload imbalance between router threads and core threads, workload balance should be achieved by migrating cores to router threads other than routers to core threads. Fig. 5(c) shows an example where a core

migrates to a router thread. This decision still obeys constraint 2 that the network must be simulated by router threads.

4. **Synchronization between router threads must be strict to increase accuracy.** Since RR link is highly-interactive, if synchronization between router threads is relaxed, the clock-skew between router threads will easily produce timing errors. So, for the sake of accuracy, router threads must synchronize very strictly. In our CRAW/P scheme, the router thread quantum is set as 1 by default. The small router thread quantum requires router threads to frequently synchronize with each other. However, the synchronization overhead is not much because the number of router thread is small.
5. **Synchronization between core threads and router threads should be relaxed to a reasonable extent to enable speedup.** The only synchronization requirement for simulating cores derives from the CR interaction. The CR links, however, are observed to be less interactive links. So, relaxing the core-core and core-router synchronization to a reasonable extent will improve the simulation speed at the expense of a moderate accuracy loss.

In summary, compared to a Static scheme, CRAW/P should present a better speedup and accuracy: Simulation speedup comes from 1) workload balancing between host threads and 2) synchronization relaxation between core threads. The accuracy improvement comes from several features: 1) it partitions the workload adaptively, achieving accuracy improvements from the workload balance; 2) it requires the network to be simulated by dedicated router threads, which limits the number of host threads that simulate the network; 3) it requires tight synchronization between router threads (small router thread quantum).

## 4 Results

This section presents the evaluation results that demonstrate the simulation speed and accuracy of our partition scheme compared to our two references. Section 4.1 describes the host and target configurations. Section 4.2 compares the performance of Static, Simple-Adaptive, and CRAW/P.

### 4.1 Experimental Setup

The multi-core host has four quad-core Intel(R) Xeon(R) E7420 CPUs running at 2.13GHZ and 128GB of DRAM. The OS is Red Hat SMP Linux with kernel version 2.6.18. Each of the experiments in this section uses the target architecture parameters summarized in Table 1 unless otherwise noted. The 256-core many-core architecture is similar with that in Fig. 1(b). The cache coherency used in the target is directory based MESI, which is similar with that of TilePro [3]. We ported representative applications from the SPLASH2 benchmark suite. Table 2 lists the problem size of these applications. All applications are threaded with 256 threads.

**Table 1.** Parameters of Target Architecture

| Feature | Value |
|---|---|
| Clock | 1GHz |
| L1 Cache | Private, 32 KB, 32-byte line size, 4-way associative, and LRU replacement. |
| L2 Cache | Shared, 128KB, 64-byte line size, 8-way associative, and LRU replacement. |
| Coherence | Directory based MESI. |
| DRAM | 64GB/s = (8 Controllers * 8 GB/s each). |
| Interconnect | Mesh network 16x16; wormhole routing; |

**Table 2.** Parameters of Target Applications

| application | Problem size |
|---|---|
| fft | 64M points |
| radix | 256M keys |
| lu | 1024x1024 |
| fmm | 4k |
| barnes | 2048 |
| cholesky | Input set tk15.O |

The experiment platform is a quantum [13] based PDES simulator—QMill. QMill derives from the GAS [22] simulator, which is an accurate simulator for the Godson-T Many-core architecture [5]. We use QMill because it can conveniently simulate hundred-core general purpose manycore architectures described in fig. 1(b).

## 4.2 Simulation Performance

**Case Study.** Fig. 7 (abc) illustrates the average synchronization overhead and timing error of the three partition schemes during a simulation run. The adaptive schemes (Simple-Adaptive and CRAW/P) largely reduce the synchronization overhead from ~20% to <10%. Meanwhile, the deviation is reduced from ~23% to <5%. Note that CRAW/P is slightly worse than Simple-Adaptive in synchronization reduction. This is because of the higher partitioning overhead and extra synchronization overhead introduced by the stricter inter-router synchronization. Fig. 7 (efg) shows the simulation speed and timing error of three schemes during simulation. Simple-Adaptive improves the speed from ~0.011MCPS to >0.015 MCPS. CRAW/P

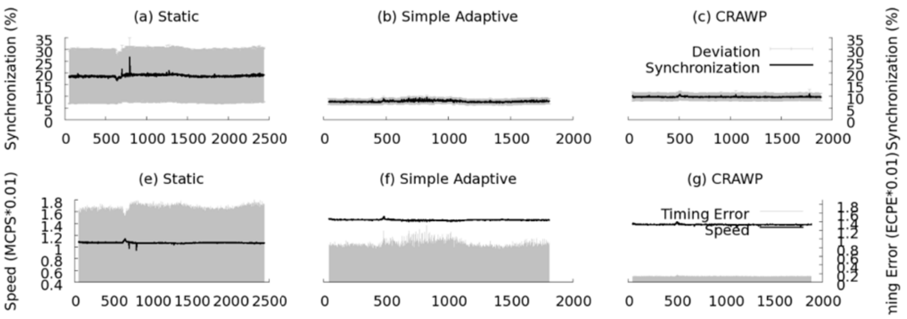


**Fig. 7.** The top graphs show the average synchronization overhead and the deviation (workload imbalance) for Static (a), Simple-Adaptive (b), and CRAW/P (c). The bottom graphs show the simulation speed (MCPS) and timing error (ECPE) for Static (e), Simple-Adaptive (f), and CRAW/P (g). Data are collected from the *matrix_multiply* kernel simulated by 8 host threads. For Static and Simple-Adaptive, the synchronization quantum is set to 8. For CRAW/P, the core thread quantum is 8 and the router thread quantum is 1.

improves the speed to >0.014 MCPS, which is slightly less than that achieved by Simple-Adaptive. However, CRAW/P reduces the timing error (ECPE) from ~0.016 (Static) to <0.002, which is far less than ~0.01 of Simple-Adaptive.

**Simulation Speed.** Fig. 8 shows the simulation speedup over the SPLASH2 benchmarks for Static, Simple-Adaptive, and CRAW/P. All applications exhibit better speed when using Simple-Adaptive or CRAW/P. The improvement of Simple-Adaptive ranges from a factor 1.37 (*choleskey_p16*) to 1.74 (*radix_p4*). The improvement of CRAW/P ranges from a factor 1.28 (*cholesky_p16*) to 1.67 (*radix_p4*).

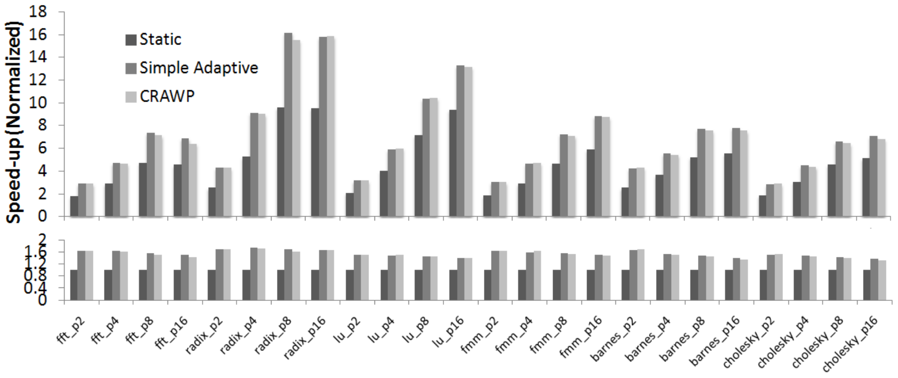
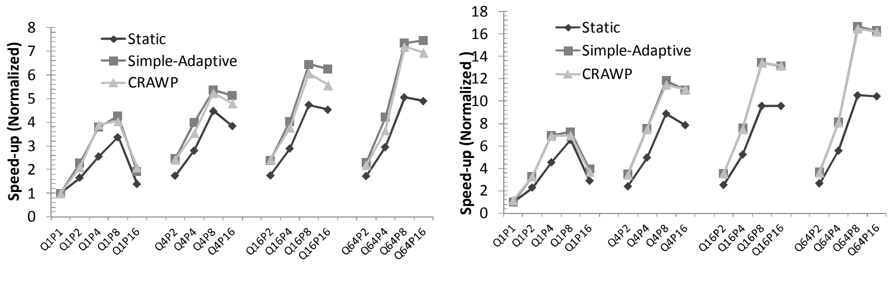


**Fig. 8.** Scaling of SPLASH benchmarks across various core counts using the three partition schemes. The top graph shows the speed-up normalized to a single core (sequential simulation). The bottom shows the speed improvement of Simple-Adaptive and CRAW/P compared to Static. The number of cores in the host is denoted by *_p*. In all simulations, the quantum size is 8 but in CRAW/P router thread quantum is 1. The number of router threads in all the simulations falls in the region of 1–2.3.



(a) *fft* (b) *radix*

**Fig. 9.** Simulation speed of (a) fft and (b) radix with different quantum sizes and different numbers of host threads. The quantum size is denoted by Q*. The thread count is denoted by P*. For example, Q4P8 indicates that the simulation is run on 8 host threads with a synchronization quantum of 4.

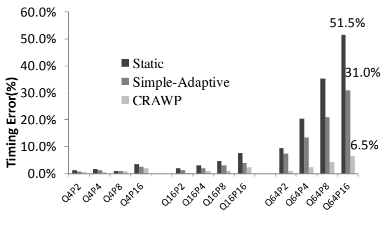
It is notable that Simple-Adaptive behaves slightly better than CRAW/P. The difference is more obvious when more host threads are involved. That happens because CRAW/P introduces more partitioning overhead because it has to consider the workload of core and router respectively. Besides, the router threads in CRAW/P synchronize in every cycle while in Simple-Adaptive the synchronization between all host threads is relaxed. The extra synchronization slows down the simulation speed of CRAW/P but warrants better accuracy.

Another observation in is that the speed improvements of Simple-Adaptive and CRAW/P generally slowly drop as more host threads are added. The observation corresponds to the growing synchronization overhead between host threads. The benefit of workload balancing is gradually reduced as synchronization dominates the simulation overhead.

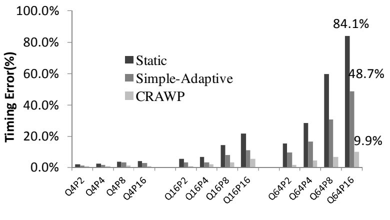
To demonstrate the effect of Simple-Adaptive and CRAW/P using different quantum, the result of *fft* (the application displaying the worst scaling) and *radix* (the application with best scaling) are selected and illustrated in Fig. 9. A general trend can be observed: adaptive partition schemes provide more improvement in cases of larger quantum sizes. The reason is the following: in quantum simulation, each host thread advances quantum cycles without synchronization, and the imbalance in each quantum is accumulated. The imbalance accumulation will be more significant as the quantum grows. A bigger workload imbalance will give more chances to adaptive partition schemes to do workload balancing.

**Simulation Accuracy.** Fig. 10 shows timing errors of *fft* and *radix* with different quantum sizes and different numbers of host threads. The timing error referred here is obtained by comparing the timing result with sequential simulation. The first fact to observe is that the timing error of *fft* is generally less than *radix*. This matches the application behavior of *fft* that contains a considerable sequential phase (the sequential phase is sequentially simulated and thus involves no timing error).

As illustrated, in the case of the Static scheme, the timing error grows intensely with 1) more host threads and 2) larger quantum. The simulation is particularly inaccurate with large quantum. The timing error in Q64P8 and Q64P16 exceeds 50% (51.5% in *fft* and 84.1% in *radix*), which makes the results simply worthless. Another notable fact is that timing error grows very significantly from Q16 to Q64. That's because a quantum of 64 would more easily yield clock skew of tens of cycles, which would completely cover the major part of idle-cycle distribution (cf. Fig. 4).



(a) *fft*



(b) *radix*

**Fig. 10.** Simulation error of (a) fft and (b) radix in simulation with different quanta across various numbers of host threads. Quantum size is denoted by Q*. Thread number is denoted by P*.

Simple-Adaptive does indeed reduce the timing error, but the saving is very limited. Although it shows better efficiency in cases with more host threads and larger quantum size, the timing error is still more than half that of Static. CRAW/P is very effective in improving accuracy, especially in the cases of Q64*. It reduces the timing error of *fft_Q64P16* from 51.5% to 6.5%, and reduces the timing error of *radix_Q64P16* from 84.1% to 9.9%. Overall, the experiment shows that CRAW/P behaves much better than both Simple-Adaptive and Static.

## 5 Related Work

Simulation is an important technique to explore new computer architectures ranging from micro-processors to parallel computers. A variety of different simulators exist, most of which are sequential. Sequential simulators run on one host thread, which limits performance. Various techniques have been studied to accelerate simulation speed including parallel simulation, direct execution, and FPGA acceleration et.

The best known parallel simulation method, PDES has been studied for decades. In conventional PDES simulators, host threads must synchronize frequently to maintain the fidelity of the simulation. Some PDES simulators adopted Quantum [13] or Slack to relax the synchronization condition.

Static workload partition is widely used in state-of-art parallel simulators. Typical examples are P-Mambo, SlackSim, and Graphite. Parallel Mambo [16] (P-Mambo) is a multi-threaded implementation of Mambo where a core based module partition is proposed to achieve high inter-scheduler parallelism. However, the evaluation only simulates a relatively small 4-core PowerPC machine.

In SlackSim, there are two types of host threads: core thread and manager thread. One dedicated thread simulates the centralized lower memory hierarchy while another set of threads (four in the paper) simulate the cores. The workload imbalance between core and memory threads can be statically avoided and the minimum L2$ access latency can be identified as safe quantum (SlackSim).

Graphite [15] uses multi-machine distributed simulation, which provides a better scalability. The tiled multicore architecture is very similar to the manycore architecture proposed in this paper. However, the workload partition is still static, with each host process simulating a set of target tiles, whose number is limited to 32.

## 6 Conclusion and Future Work

This paper addresses workload partitioning in manycore simulation. We discuss the architectural characteristics of manycores, present the drawbacks of a static scheme for manycore simulation, and propose an adaptive workload partition method called CRAW/P. Experimental results demonstrate that CRAW/P delivers considerable speedup (28–67%) and accuracy saving (<10% in timing error with a quantum of 64).

Further digging into the mechanisms lying behind the effects of workload imbalance and network on speed and accuracy can help us better understand manycore simulation; it can also provide future improvement opportunities. These opportunities are the focus of our future work. Another possible extension is to partition the simulation of cores and network onto different machines, seeking better performance to simulate large-scale manycore architecture containing more cores.

**Acknowledgment.** This work is in part supported by the National Grand Fundamental Research 973 Program of China under Grant No. 2011CB302501, the National Science Foundation for Distinguished Young Scholars of China under Grant No. 60925009, the Foundation for Innovative Research Groups of the National Natural Science Foundation of China under Grant No. 60921002, the Beijing science and technology plans under Grant No.2010B058 and the National Natural Science Foundation of China under Grant No.(61173007， 61100013 and 61100015).

## References

[1] Howard, J., Dighe, S., Hoskote, Y., et al.: A 48-Core IA-32 Message-Passing Processor with DVFS in 45nm CMOS. In: Proceedings of the International Solid-State Circuits Conference, ISSCC 2010 (February 2010)

[2] Vangal, S., et al.: An 80-Tile 1.28 TFLOPS Network-on-Chip in 65nm CMOS. In: IEEE International Solid-State Circuits Conference, ISSCC 2007. Digest of Technical Papers, pp. 98–589 (2007)

[3] Bell, S., et al.: TILE64 processor: A 64-core SoC with mesh interconnect. In: Proceedings of the International Solid-State Circuits Conference, ISSCC 2008 (February 2008)

[4] The TILE-GxTM Processor Family, Tilera (2009), `http://www.tilera.com/products/processors`

[5] Fan, D., Zhang, H., Wang, D., et al.: High-Efficient Architecture of Godson-T Many-Core Processor. In: Proceedings of Hot Chips 23. IEEE Computer Society (2011)

[6] Kelm, J.H., Johnson, D.R., Johnson, M.R., et al.: Rigel: An Architecture and Scalable Programming. In: ISCA 2009 (2009)

[7] Burger, D., Austin, T.: The SimpleScalar tool set, version 2.0. Technical Report TR-1342, University of Wisconsin-Madison Computer Sciences Department (June 1997)

[8] Binkert, N.L., Dreslinski, R.G., Hsu, L.R., Lim, K.T., Saidi, A.G., Reinhardt, S.K.: The M5 Simulator: Modeling Networked Systems. IEEE Micro 26, 4 (2006)

[9] Magnusson, P.S., et al.: Simics: A Full System Simulation Platform. IEEE Computer 35(2), 50–58 (2002)

[10] Chidester, M.C., George, A.D.: Parallel simulation of chip-multiprocessor architectures. Proceedings of ACM Trans. Model. Comput. Simul., 176–200 (2002)

[11] Steinman, J.S.: SPEEDES: A Multiple-Synchronization Environment for Parallel Discrete-Event Simulation. International Journal in Computer Simulation 2, 251–286 (1992)

[12] Chandy, K.: Distributed Simulation: A Case Study in Design and Verification of Distributed Programs. IEEE Transactions on Software Engineering 5(5), 440–452 (1979)

[13] Mukherjee, S.S., Reinhardt, S.: Wisconsin Wind Tunnel II: A Fast, Portable Parallel Architecture Simulator. IEEE Concurrency 8(4), 12–20 (2000)

[14] Chen, J., Annavaram, M., Dubois, M.: SlackSim: A Platform for Parallel Simulations of CMPs on CMPs. SIGARCH Comput. Archit. News 37(2), 20–29 (2009)

[15] Miller, J.E.: Graphite: A distributed parallel simulator for multicores. In: HPCA 2010: The 16th IEEE International Symposium on High-Performance Computer Architecture (2010)

[16] Wang, K., Zhang, Y., Wang, H., Shen, X.: Parallelization of IBM mambo system simulator in functional modes. Operating Systems Review, 71–76 (2008)

[17] Chiou, D., Sunwoo, D.: FPGA-Accelerated Simulation Technologies (FAST): Fast, Full-System, Cycle Accurate Simulators. In: MICRO 2007: Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 249–261 (2007)
[18] Chung, E.S., Papamichael, M.K., Nurvitadhi, E., Hoe, J.C., Mai, K., Falsa, B.: ProtoFlex: Towards Scalable, Full System Multiprocessor Simulations Using FPGAs. ACM Trans. Recongurable Technol. Syst. 2(2), 1–32 (2009)
[19] Dave, N.: Implementing a functional/timing partitioned microprocessor simulator with an FPGA. In: 2nd Workshop on Architecture Research using FPGA Platforms, WARFP 2006 (February 2006)
[20] Monchiero, M., Ahn, J.H., Falcon, A., Ortega, D., Faraboschi, P.: How to simulate 1000 cores. SIGARCH Comput. Archit. News 37(2), 10–19 (2009)
[21] Dybdahl, H.: An Adaptive Shared/Private NUCA Cache Partioning Scheme for Chip Multiprocessors. In: Proc. of the Int. Symposium on High Performance Architecture (HPCA), pp. 2–12 (2007)
[22] Huiwei, L., et al.: P-GAS: Parallelizing a Cycle-Accurate Event-Driven Many-Core Processor Simulator Using Parallel Discrete Event Simulation. In: 24th ACM/IEEE/SCS Workshop on Principle of Advanced and Distributed Simulation (PADS 2010), Atlanta, USA (June 2010)
[23] Jefferson, D., Beckman, B., Wieland, F., Blume, L., Diloreto, M.: Time warp operating system. In: Proceedings of the 11th ACM Symposium on Operating System Principles, pp. 77–93 (1987)
[24] Das, S.R., Fujimoto, R., Panesar, K.S., Allison, D., Hybinette, M.: GTW: a time warp system for shared memory multiprocessors. In: Winter Simulation Conference, pp. 1332–1339 (1994)