A Novel Memory Subsystem Evaluation Framework for Chip Multiprocessors

Fucen Zeng, Lin Qiao, Mingliang Liu, and Zhizhong Tang
Department of Computer Science and Technology
Tsinghua University
Beijing, China
Email: zengfucen@gmail.com, qiaolin@tsinghua.edu.cn, liuml07@gmail.com, tzz-dcs@tsinghua.edu.cn

***Abstract*—This paper presents a fast and cycle-accurate memory subsystem modeling and evaluating framework for Chip Multiprocessors (CMPs), called TSIM (Tsinghua SIMulator), which gives a flexible and extensible approach to evaluating architecture designs, models or algorithms, including the network-on-chip interconnection, cache hardware prefetcher, memory system protocol, replacement policy, *etc*. TSIM is trace-driven, adopting a dynamic binary instrumentation technique to generate the running trace information of applications on-the-fly. After receiving the trace information, TSIM will reappear the on-chip memory behaviors of applications. By introducing the concept of statistical meta metrics, TSIM separates the analysis stage from the simulation process *per se*, and this provides a great facilitation for a user to count and sample the performance metrics. Compared to the real cache system, TSIM achieves an accuracy of 90.66% at the average speed of 327 KIPS. Meanwhile, TSIM accelerates the simulating speed by 10 to 100 times, compared to the traditional cycle-accurate cache simulators. On the other hand, when TSIM is used to characterize the on-chip memory system behaviors of SPEC CPU 2000 benchmarks, experimental results about the on-chip memory behaviors are the same as others.**

I. Introduction

With more and more cores packaged in a Chip Multiprocessor (CMP), memory hierarchy is becoming one of the key design problems. But unfortunately, for almost all of computer architectures, quantitative evaluation of memory subsystems is possibly only by using simulators [1]. In other words, simulation is indispensable for a computer architect. Yet even as many simulators and other simulating methods have been presented for CMP researchers, to date there is still some main problems remained unsolved.

One such a problem is unbearable simulating speed. It is virtually impossible to simulate all benchmarks of a whole suite to completion, especially by using those full-system simulators [1]. Typically, an architect simulates a subset of benchmarks, using a reduced or truncated input set or a representative set instead, to minimize the simulating time. As a result of trading speed for accuracy, these simulators bring out the second problem: poor accuracy, while an evaluation with poor accuracy is untrusted. Besides, these simulators are often hard to use or to implement new architecture solutions.

This work is supported by the National High Technology Research and Development Program of China (863 Program) under Grant No. 2008AA01Z108, and National Science Foundation under Grand No. 60773149.

To address these problems, this paper proposes a fast and cycle accurate memory subsystem evaluation framework, called TSIM (Tsinghua SIMulator). TSIM focuses on CMP systems and supports multithread workloads. As a sequel to [2] and [3], TSIM uses a dynamic instrumentation tool, pin, as its front-end to generate trace files of running workloads. After permissible reduction, a reduced trace is sent into the simulation core to generate performance statistical result for a specific architecture. Then, a user can use some post-processing tools to parse the statistical result and to analyze the performance of the workload. The statistics and sampling technique in TSIM, based on the statistical meta metrics (SMM), makes a user totally free to get any performance metrics. More significantly, TSIM presents an extensible approach to exploring behaviors of on-chip subsystems. A TSIM user can configure simulation parameters freely, such as cache level, cache size, block size, number of cores, cache associativity, cache resources, hit latency, replacement policy and cache protocol.

Unlike other simulation methods, TSIM has been trying to find the right balance among speed, accuracy and flexibility. By simulating the delays caused by various types of events, TSIM performs cycle-detailed simulation, and this ensures the accuracy of the simulation results. At the same time, by simplifying pipelines and highly optimizing codes, TSIM also provides relatively higher simulating speed with little accuracy loss. Compared to some traditional simulatiors, TSIM accelerates the memory subsystem simulation speed by 10 to 100 times, with only 9.34% accuracy loss.

This paper is organized as follows. Section II discusses related work. Motivation and mechanism of TSIM are reported in Section III, while design and implementation of TSIM are further elaborated in Section IV. Section V provides experimental results and Section VI concludes.

II. Related Work

During the past decades, many simulators have been used in architecture performance evaluation and optimization field. Practically, there are three main types of them: execution-driven simulation, instrumentation-driven simulation and trace-driven simulation.

Execution-driven simulation relies on existing functional performance models to execute a binary application [4], [5],

[6], [7]. The functional performance model provides the memory addresses in real time. A cache simulation model starts after receiving the access information. GEMS [4], as a simulation tool set, is a typical case of execution-driven simulator, created by Wisconsin Multifacet Project, to characterize and evaluate the performance of multiprocessor hardware systems. Fahringer *et al.* [5] have used an execution-driven model to analyze performances of distributed and parallel systems. Woo *et al.* [6] have also used an execution-driven simulating model with the Tango Lite Tracing tool [7].

An instrumentation-driven simulator uses a dynamic binary instrumentation tool to gather the running information of an application at any expected instrumented point [8], [9]. Then, after receiving the running information such as memory addresses, the cache model works. A binary instrumentation approach is relatively faster, suitable for conducting accurate memory performance studies. Jaleel *et al.* [8] have described an instrumentation-based approach to characterize memory behaviors of workloads. They use CMP$im, an instrumentation-driven memory simulator, to evaluate the memory performance. Since binary instrumentation normally occurs at native execution speed, an instrumentation-driven simulator can run at MIPS (Million Instructions Per Second) speed. However, they do not simulate a detailed timing model, and do not provide a prefetcher either. Above all, CMP$im is not available to public.

A trace-driven simulator uses some specific tool(s) to collect the memory references (called an address trace) of a running application, and applies the trace sequence to the simulation model to mimic the way that a real processor might exercise the design [10], [11]. A trace-based simulation method is conceptually simple, and easy to reappear experimental results. Since functional models of modern ISAs are considerable slow and complicated to be built. Trace-driven simulation is more popular for conducting memory subsystem performance studies [10], such as the performance characterization and performance optimization. Uhlig *et al.* [10] have made a detailed survey for existing trace-driven simulators. They have investigated more than 50 trace-driven simulators and showed that none of them is best when all criteria (including accuracy, speed, memory, flexibility, portability, ease-of-use, etc.) are considered together. SimpleScalar [11] is one of the hugely popular set of simulation tools during the past 15 years. But in this multicore era, on-chip cache access model cannot be applied to it directly.

Other researchers have also proposed simulation environments to meet their own research needs. Li *et al.* [12] use IBM's TurandotPowerTimer to generate single-core L2 cache-access traces that are annotated with timestamps and power values, then feed these traces to a cache simulator developed by themselves. The simulator uses hits and misses to shift the time and power values in the original traces. They propose joint optimization across multiple design variables. Bononi *et al.* [13] use OMNeT++ simulation Framework to analysis some architectures for network on chip. Sun *et al.* [14] construct a prototype using a public domain network simulator ns-2 and evaluated design options for a specific network-on-chip (NoC) architecture.

In summary, to our knowledge, few researchers have comprehensively considered many cores and mutual effects of timing components under the CMP environment; namely, CMP researchers still lack of a useful timing-detailed memory subsystem performance model to support the x86 CMP platform. Some full system simulators such as SIMICS [15] and M5 [16] do support x86 ISA, but they are bulky and emulate the whole system with peripherals and the operating system: apt to be accurate but much slower. It is expected that this paper fills the gap.

III. Framework Methodology

As a cycle-accurate memory subsystem performance evaluation framework, TSIM is based on both of a trace-driven technique and a dynamic binary instrumentation one. TSIM separates the hardware timing logics into several flexible modules, provides a unified interactive simulation method under the same timing scheduling. Simulation of many cores has also been supported in the TSIM framework.

There are three main stages in the TSIM framework. First, TSIM collects a trace of a running workload in the front-end stage. Second, in the back-end stage the reduced trace will be sent into the simulation core to generate performance statistical result for a specific architecture. In order to decrease storage cost and to shorten simulation time, permissible reduction may be made without losing accuracy. Last, in the analysis stage a user can use some post-processing tools to parse the statistical result and analyze the performance of the workload. TSIM is, in fact, a fast and efficient tool for researchers who are interested in CMP on-chip memory subsystems.

A. Front-end Basis

TSIM adopts a binary instrumentation tool, pin, as its front-end. Pin [17] is a well known dynamic instrumentation tool for Linux and Windows binaries, which supports the instruction set of IA32, IA32E, ARM, and Itanium processors. Similar to the ATOM [18], Pin can instrument at any point of the target application, by using the just-in-time compiling technique to insert and optimize codes.

In addition to those common optimization techniques in a dynamic instrumentation system, including code caching and trace linking, Pin employs lots of other techniques to optimize object codes compiled on-the-fly, such as register reallocation, inlining, liveness analysis and instruction scheduling. Pin also supports a certain degree of transparency, which ensures the completeness and correctness of the trace information collected by the instrumentation tool.

As a trace generator, the front-end module uses the APIs of Pin to interact with binary code, collects the trace of the running application, and then tells Pin to control the execution of the application if necessary. In order to provide the memory access information to the TSIM back-end modules, the trace generator mainly collects the memory references and the information of the memory accessing instructions. As the

input of the whole simulation process, each record in the trace contains thread id, accessed memory address, data size, memory access type (read or write), and so on.

B. The Structure of the Target Memory Subsystem

Figure 1 shows the structure of the target memory subsystem modeled in TSIM. Caches or other objects on chip are connected through the interconnection network.

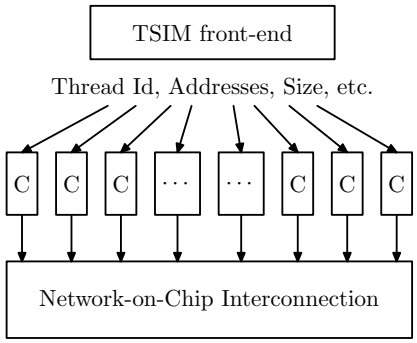

Fig. 1. The structure of the target memory subsystem in TSIM

Several components have already been built in the TSIM framework as fixed modules that can be assembled and configured but not removed (*e.g.*, Cache Module and Pipeline Module), while other modules as flexible ones, such as Network-on-Chip Interconnection (NCI) Module, Cache Prefetcher (CP) Module, Replacement Policy (RP) Module, Memory System Protocol (MSP) Module. In the TSIM framework these flexible modules are extensible: a user can not only assemble or configure the existing implementation of these modules but also replace a module with a new one.

While complicated out-of-order pipelines have been a hot point for decades, recent studies [3] have shown that simple in-order cores would be the next attractive resort for CMPs. Thus, the TSIM adopts simplified pipelines to design the detail of the memory model, which brings little accuracy loss. Also, it is easy for TSIM to be extended under more detailed investigation.

IV. Design and Implementation

The TSIM models the hardware timing of the processor and on-chip memory subsystem. Also, it constructs uppermost abstraction layers of common research hot-points, such as network-on-chip interconnection, cache prefecther, cache coherence policy and replacement policy. Since the statistical strategy has already been built in the uppermost abstraction layer of each module, the performance statistics can be completed automatically when a TSIM user implements a new algorithm as a substitute for a predefined one. It is flexible enough for the TSIM user to make changes of modules to implement new improvements and innovations.

A. Network-on-Chip Interconnection

The NCI module models performance effects caused by network-on-chip interconnection of memory sub-systems on CMPs. Two major characteristics are modeled: network structure (topology) and latency of messaging.

Figure 2 shows the structure of the NCI module in TSIM. Every component on chip, such as data cache (DC), instruction cache (IC) of each core and other logic objects, can interconnect with each other via the NCI module. The NCI module is responsible for creating a logical network interface card (NIC) for those objects connected to the NCI, while an NIC is in charge of packaging, sending, and receiving packages for the corresponding node.

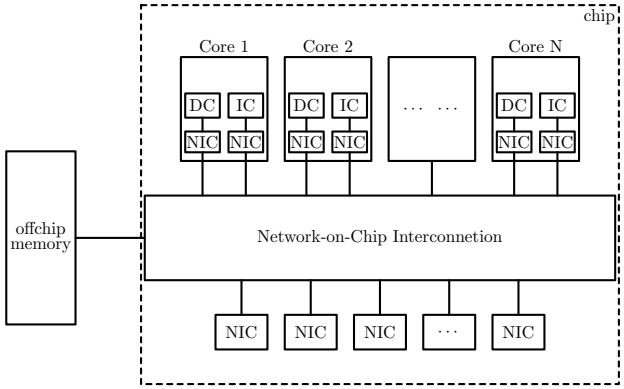

Fig. 2. The Structure of the NCI in the TSIM framework

The TSIM provides a flexible and extensible support for the NCI research. Not only those components on chip can connect to the NCI module, but also some off-chip components can do either; for example, the off-chip memory, as shown in the figure 2. Basically, there is no limits for the number of the network nodes, if only the memory of the host machine permits.

The NCI module simulates details of real packet transfer processes with a routing algorithm, or just facilitates the processes by directly using the average transfer cycles to complete the packet transfer simulation. Both of the average throughput of network and the throughput of each network node are obtained.

In addition, Wang *et al.* have investigated various on chip interconnection network structures in CMPs [19]. Their research has suggested that for a small-scale and low-load op-chip system a multi-ring network works well, but for a very-large-scale on-chip system it is very difficult to get good performance by just adopting any NoC interconnection structure though a mesh grid is better than a ring one. That is, strictly speaking, how to choose a proper NoC interconnection structure is still open. TSIM displays a possibility, an opportunity of making researches on NoC interconnection. Both of mesh and a bus-based interconnect network have already been implemented in the code repository, while networks based on others next.

B. Cache Prefetcher

On-chip cache prefetching operations may be at any access or just when access-miss occurs. It is a very important way to optimize the on-chip cache performance. The TSIM provides

an independent module to satisfy the needs of cache hardware-prefetcher researches.

For the CP module, two operations are untrivial: *Notify* and *SendPrefetchRequest*. Memory access information at every cycle is passed to the prefetcher module via the *Notify* operation. Then, the prefetcher instance updates its internal data structure and calculates the prefetch-packets. At the same time, the prefetcher instance sends a request to the targeted cache to load the prefetch via the *SendPrefetchRequest* operation if conditions are satisfied.

In the current version of TSIM several cache hardware-prefetcher algorithms have been implemented:

- A sequential prefetcher, which just prefetches the next block at the next cycle when a miss happens [20],
- Global History Buffer Prefetcher (GHB), which prefetches based on the information recorded in a global history buffer [21],
- Adaptive CZone/Delta Correlations Prefetcher (AC/DC), based on GHB prefetcher, which divides the memory address space into equal-size concentration zones (CZones) and uses a global history buffer to track and detect patterns in miss address deltas (differences between consecutive addresses) within each CZone [22],
- Markov Prefetcher, which makes use of the miss address stream as the prediction source and models the miss address stream as a Markov graph: informally, a probabilistic state machine [23],
- Stride Prefetcher, which depends on stride distance and degree, where the stride distance is the distance of two consecutive memory addresses and the degree the number of prefetch requests issued one time [24], and
- Tag Correlating Prefetcher (TCP), which works with tags instead of cache-block addresses is significantly more resource-efficient [25].

C. Replacement Policy

Replacement policy is one of the main optimization targets to achieve high performance. In TSIM, the uppermost abstraction has been given, accompanied by several existing replacement policies, such as LRU and Random replacement policy.

The RP module works when the cache has received a memory request. A user who wants to evaluate a replacement algorithm just needs to implement only one operation to access the cache. This operation is responsible for accessing the cache set, updating it and returning the access result or a victim cache block if a miss occurs.

D. Memory System Protocol

In TSIM, memory system protocol only means coherence protocol, but actually the cache control logic, which defines various status transitions based cache control behaviors including cache coherence.

To implement a new memory system protocol in TSIM, a user has to define the cache block statuses used in the protocol and the status transitions within those status classes. When a protocol is instantiated, a set of status objects (*e.g.*, *Modified*, *Shared* and *Invalid*) is created.

TSIM now supports two protocols: MESI (known also as *Illinois* protocol due to its development at the University of Illinois at Urbana-Champaign) [26] and a simple invalidate-based cache coherence protocol. The support of MOESI based cache coherence protocol is part of the on-going work. These protocols work with the network interconnection which is simulated in the NCI module. All of coherence behaviors, cache controlling commands and network traffic caused by these operations can be collected in TSIM.

E. Statistics and Sampling

The task of the statistics and sampling (SS) module is to provide a way to collect or sample statistical information and then to print them out in a certain format. In TSIM, the principle of recording statistical information is that only basic accumulative statistical data which can be sampled from hardware event counter (*e.g.* number of L1D cache access) are maintained, while others that can be derived from the basic statistical data are discarded. As described above, these basic statistical data are called SMMs.

A TSIM user can get any performance metrics from these SMMs. For example, the number of the cache access hits (CAH) and the number of cache access (CA) are two SMMs, a user can get the miss-rate via $(CA - CAH)/CA$. In this way, TSIM treats the statistical recording process and analysis process as two independent functions.

Furthermore, in order to be extensible enough, the statistical data are organized as a tree, which can be dumped out to a output stream in an XML file. It is very convenient for a TSIM user to gather the statistical meta data with an XML parser or a shell script in the analysis stage. It can also be done if extra computation of the performance metrics is needed.

The TSIM sampling tool also allows a user to get a general view on the data changes of statistical meta nodes. Sometimes, the changing information of the statistical meta node is very useful to research into application behaviors under the on-chip memory environment. Consequentially, however, additional huge I/O operations slowdown the simulating speed. Hence, it is suggested that the sampling interval should be larger than 10,000 cycles.

V. Experimental Evaluation

This section evaluates TSIM by using SPEC2000 and NPB-OMP benchmarks, which are representative of a large subset of the benchmarks normally used in the computer architecture literature.

A. Experimental Testbeds

All experiments run at a 4-way SMP system, and the operating system is linux. The detailed experimental configuration is listed in Table I.

TSIM is used to characterize data cache (first level) behaviors of the NPB-OMP (Nas Parallel Benchmarks 3.2, OpenMP version), to give out a speed illustration. NPB is a widely used

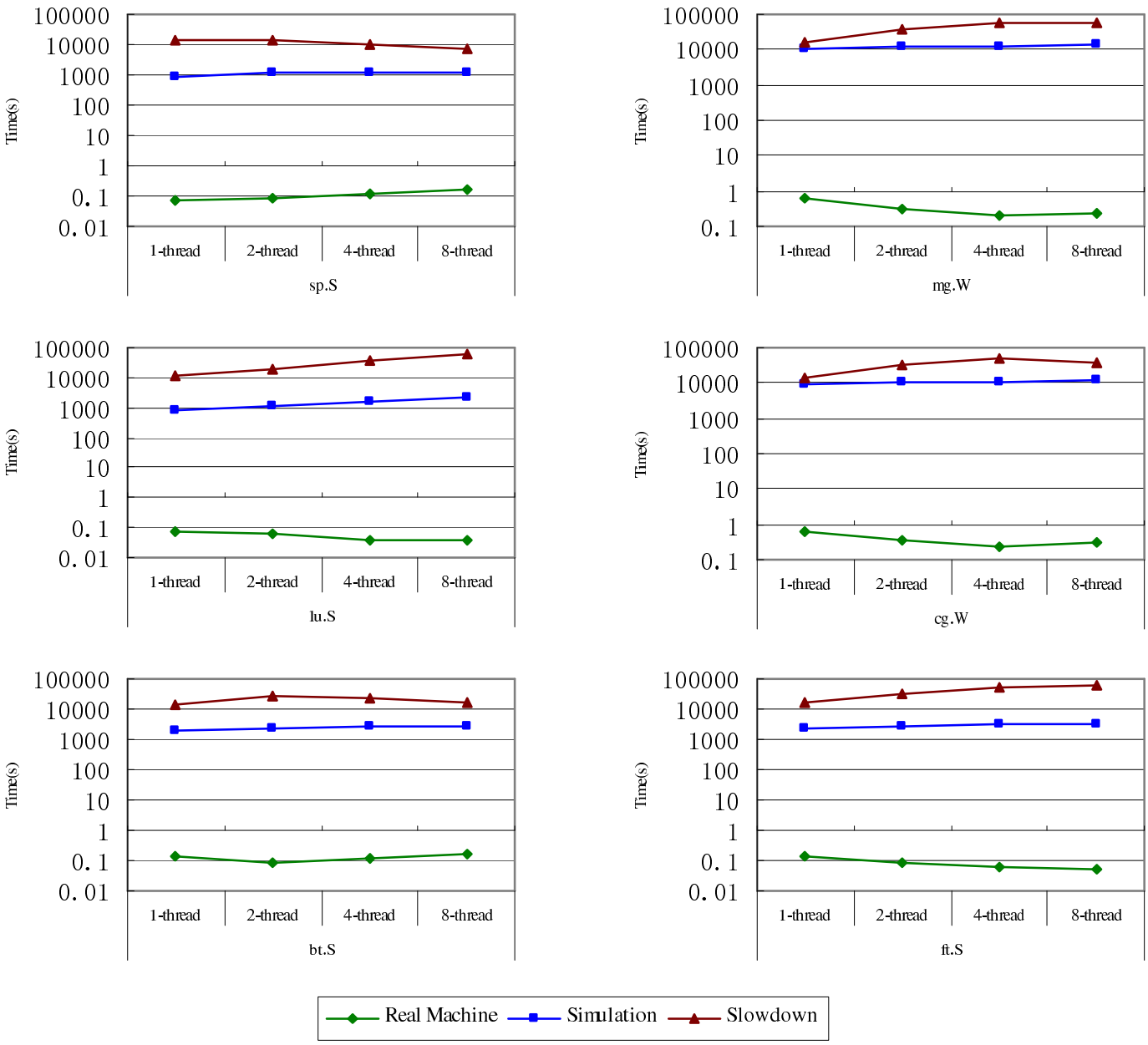

Fig. 3. The simulating time of the six NPB-OMP benchmarks on TSIM

TABLE I
THE EXPERIMENT CONFIGURATION SUMMARY

Configuration Name	Value
CPU clock frequency	2.33GHz
Cache size of host system	6144KB
Operating System	Linux-2.6.22-15
Cache Size of simulator	32KB
Cache Block Size	64B
Cache Ports	4 ports for wr
Number of MSHRs	4
Hit Latency	3 cycles
Load Latency	1 cycles
Way of Associativity	4-way
Replacement Policy	LRU

set of programs designed to help evaluate the performance of parallel supercomputers. To analysis the accuracy of TSIM, the misses per thousand instructions (MPKI) of the SPEC CPU 2000 benchmarks are compared to the value of the real machine under the same configuration.

$$MPKI = \frac{Total\ Misses}{Total\ Instructions/1000} \quad (1)$$

Finally, as an example, the TSIM sampling tool is used to get a continuous view of the miss rate of six SPEC CPU 2000 benchmarks selected randomly. In order to make the analysis brief, the simulator has been set up to skip the first one billion instructions and then to simulate the following one billion

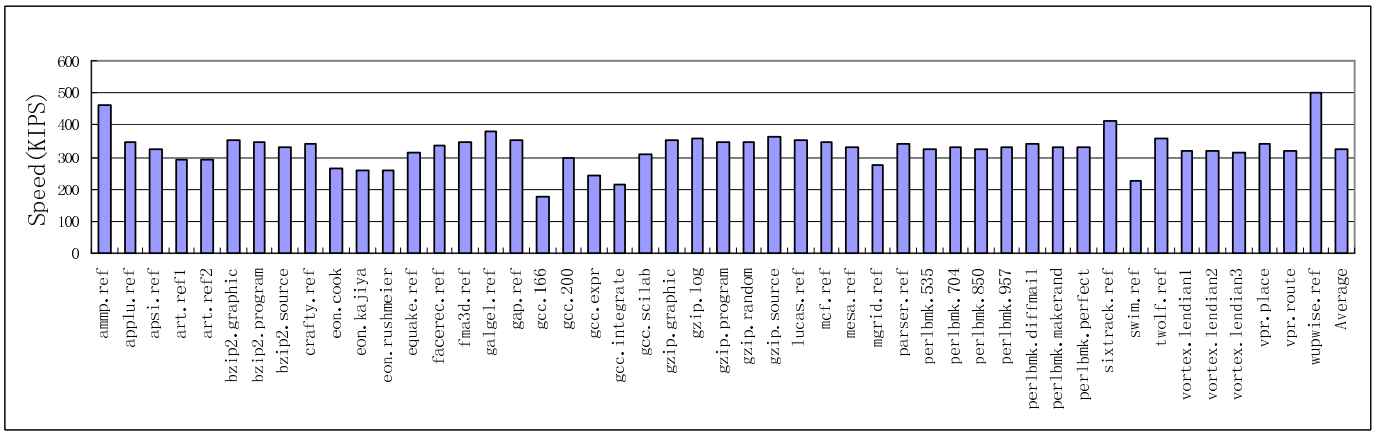

Fig. 4. Speed of simulating the SPEC CPU 2000

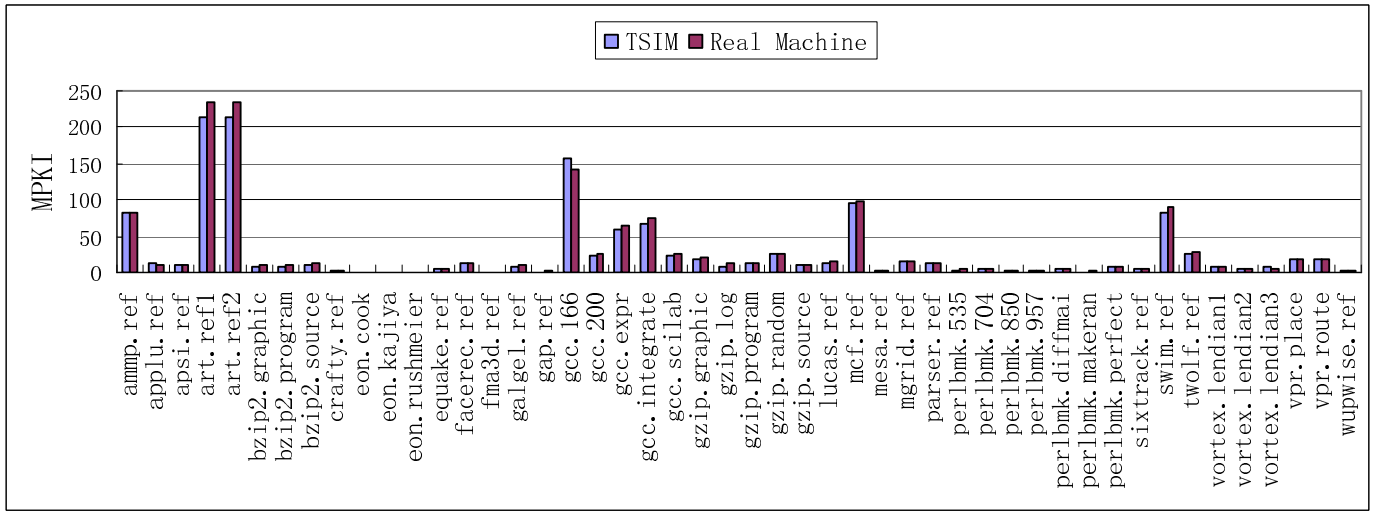

Fig. 5. MPKI comparison of TSIM and the real machine.

ones.

B. Speed and Accuracy

Figure 3 shows simulation results of the six NPB-OMP benchmarks. Each of the six programs executes three times with 1, 2, 4, and 8 threads, both on the locale machine and in TSIM, respectively. For the sake of clarity, one thread per core is assumed.

Intuitively, the rate per second at which addresses are processed can be employed to measure the speed of trace-driven simulators, but actually, it is very difficult to use this metric to compare simulators or processors that have been implemented on different hardware platforms. Because the number of addresses processed per second by a particular platform is a function of the speed of the host hardware, it is not meaningful to compare this rate to that obtained by a different processing method implemented on another host system. To overcome this difficulty, we use the term of *Slowdown*, the times that the simulator is slower than the local machine on which simulation is taken place. Mathematically, slowdown is defined as follows:

$$Slowdown = \frac{Simulation\ Time}{Host\ System\ Execution\ Time} \quad (2)$$

Thus, an approximate comparison to other methods implemented on different hosts could be made via the definition described in Equation 2. Very few papers have reported overall slowdowns because most of them tend to focus on just one aspect of trace-driven simulation, such as trace collection or the results, not the speed of their simulations. It has been shown that the speed slowdown which a common cycle-accurate simulator brings in is between 10,000 and 1,000,000 times [3]. In the experiments, TSIM is compared to traditional trace-driven simulators with this metric.

As illustrated in Figure 3, the *Native* denotes the time running on the locale machine, while the *Simulation* the time running in TSIM, and the *Slowdown* the ratio of $Simulation/Native$. Many research work were impeded by the slow cycle-accurate simulators. On the average, the slowdown which TSIM brings in is between 10,000 and

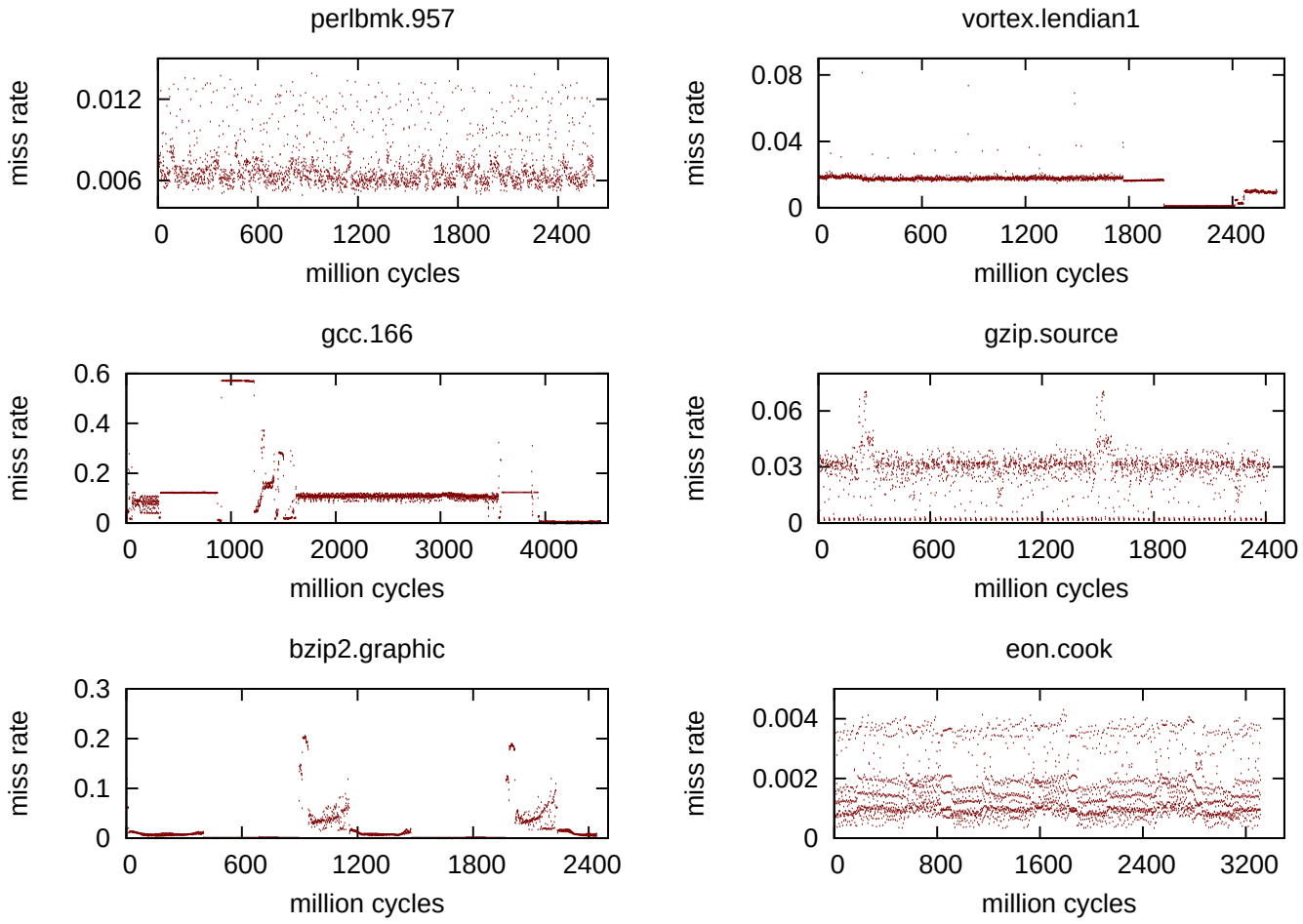

Fig. 6. Miss rate changes of the six selected SPEC CPU 2000 benchmarks running on TSIM

100,000 times. Besides, for multi-thread applications, some extra speedup can be achieved. This shows that TSIM increases the speed of cache simulation by 10 to 100 times, compared to the traditional cycle-accurate simulators, for the analyzed benchmarks.

Figure 4 shows the speed of all the SPEC CPU 2000 benchmarks running in TSIM, in thousand instructions per second (KIPS). The average speed of these 48 benchmarks is 327 KIPS, better than many cycle-detailed simulators. The *wupwise.ref* benchmark runs at the highest speed of 500 KIPS.

To evaluate the accuracy of TSIM, the MPKI of TSIM is compared to the real value of the real machine. Figure 5 shows the experimental result. It can be found that the MPKI value of TSIM is very close to the real one. And for those benchmarks whose MPKI are larger than 1, the average MPKI relative error rate is 9.34%. That is, TSIM achieves an accuracy of 90.66%.

On the other hand, there is subtle differences between the two results for the benchmarks. The reasons are, first, the number of the instructions counted by the performance monitoring tool on a real machine is different from that in TSIM, and second, the replacement policy of the real machine is not an LRU algorithm strictly, but a substitutional one, such as pseudo LRU. In general, the cache performance of TSIM is almost as same as the real cache system.

C. Detecting Performance Phase

Figure 6 shows the miss rate changes of the six selected representative benchmarks of the SPEC CPU 2000, via the TSIM's sampling tool, where a clear panorama of the workloads' cache behaviors can be observed. For example, the behavior phase of the workload *bzip2.graphic* occurs every 1,000 Million cycles in the truncated instruction interval, while *eon.cook*'s phase every 800 million cycles, *gcc.166*'s phase every 2,800 million cycles, *gzip.source*'s phase every 1,200 million cycles, *perlbmk.957*'s phase every 700 million cycles, and the *vortex.lendian1*'s phase every 700 million cycles. The rest benchmarks of the SPEC CPU 2000 are similar to these in the experiment.

Observation above is as same as the research result of Jaleel *et al.* [9]. They get the result by analyzing the miss rate changes with the number of instructions increasing, while TSIM analyzes it with the number of cycles increasing. Of course, TSIM also provides such a function to sample the miss rate changes with the number of instructions increasing.

Actually, TSIM treats these operations as a part of the analysis process since in TSIM the simulating process and simulation analysis are two functions, independent each other.

There are so many sampling statistical meta nodes, which have been supported by the TSIM's current version, that a TSIM user can get much more statistical results of the performance metrics via these statistical meta data, not limited to the miss rate, MPKI, IPC, *etc.*

VI. Conclusions and Future Work

This paper presents a fast and cycle-accurate on-chip memory subsystem performance simulation framework, TSIM. It focuses on the support of the CMP environment on x86 platform to meet the current needs. The framework is trace-driven, with the time-efficient Pin-tool-based trace generator. TSIM is cycle-detailed since it models the details of an on-chip memory subsystem. It is also easy-to-use, flexible, configurable and simple-to-modify. By introducing the concept of statistical meta metrics, the Statistics and Sampling Module of TSIM provides a great facilitation to count and sample the performance metrics.

It is shown that TSIM achieves an accuracy of 90.66%, at the average speed of 327 KIPS. It accelerates the simulating speed by 10 to 100 times, compared to the traditional cycle-accurate cache simulators.

To demonstrate the use of TSIM to characterize the on-chip memory subsystem performance, six representative SPEC CPU 2000 benchmarks have been selected randomly; and the miss rate changes with the number of cycles increasing have been sampled. The experimental results about the on-chip memory behaviors are the same as others. Besides the performance studies mentioned above, TSIM is also designed to explore the research of other directions: network-on-chip interconnection, replacement policy, cache coherence protocol, prefetching mechanism, optimization solutions for on-chip memory subsystem, *etc.*

In the future, authors are going to try to increase the speed of TSIM to MIPS (Million Instructions Per Second) and to simulate more details for on-chip memory subsystems. At the same time, it is always the on-going work to enhance the support of multithread and multiprocess applications. Last but not the least, The source code of TSIM is expected to be released in public soon.

References

[1] J. J. Yi and D. J. Lilja, "Simulation of computer architectures: Simulators, benchmarks, methodologies, and recommendations," *IEEE Transactions on Computers*, vol. 55, pp. 268–280, March 2006.

[2] M. Liu, L. Qiao, Y. Chen, F. Zeng, and C. Zhang, "An extensible memory simulation framework for chip multi-processors," in *Proceedings of the 2nd International Conference on Computer Science and Software Engineering (CSSE2009)*, 2009.

[3] Y. Chen, "Research on cache optimizations for streaming accesses and sharing behaviors on chip-multiprocessors," *Tsinghua University PhD. thesis*, 2009.

[4] M. Martin, D. J. Sorin, B. M. Beckmann, and M. R. Marty, "Multifacet's general execution-driven multiprocessor simulator(gems) toolset," *ACM SIGARCH Computer Architecture News*, vol. 33, n.4, Nov. 2005.

[5] T. Fahringer, B. Scholz, and X.-H. Sun, "Execution-driven performance analysis for distributed and parallel systems," in *In 2nd International ACM Sigmetrics Workshop on Software and Performance (WOSP 2000), Ottawa, Canada*, Sep 2000.

[6] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The splash-2 programs: Characterization and methodology considerations," in *Proceedings of the 22nd International Symposium on Computer Architecture (ISCA), Santa Margherita Ligure, Italy*, 1995.

[7] S. R. Goldschmidt and J. L. Hennessy, "The accuracy of trace-driven simulations of multiprocessors," *Tech References Rep. CSL-TR-92-546, Stanford University*, Sept. 1992.

[8] A. Jaleel, "Memory characterization of workloads using instrumentation-driven simulation - a pin-based memory characterization of the spec cpu2000 and spec cpu2006 benchmark suites," tech. rep., VSSAD, 2007.

[9] A. Jaleel, S. R. Cohn, C.-K. Luk, and B. Jacob, "Cmp$im: A binary instrumentation approach to modeling memory behavior of workloads on cmps," tech. rep., UMD-SCA, 2006.

[10] R. A. Uhlig and T. N. Mudge, "Trace-driven memory simulation: A survey," *ACM Computing Surveys*, vol. 29, 1997.

[11] T. Austin, E. Larson, and D. Ernst, "Simplescalar: An infrastructure for computer system modeling," *IEEE Computer*, vol. 35(2), 2002.

[12] Y. Li, B. Lee, D. Brooks, Z. Hu, and K. Skadron, "Cmp design space exploration subject to physical constraints," in *12th International Symposium on High Performance Computer Architecture (HPCA-12)*, 2006.

[13] L. Bononi and N. Concer, "Simulation and analysis of network on chip architecture: Ring, spidergon and 2d mesh," in *Proceedings of the conference on Design, automation and test in Europe: Designers' forum*, 2006.

[14] Y. Sun, S. Kumar, and A. Jantsch, "Simulation and evaluation for a network on chip architecture using ns-2," in *20th IEEE Norchip Conference*, 2002.

[15] P. Magnusson, M. Christensson, J. Eskilson, and D. Forsgren, "Simics: A full system simulation platform," *IEEE Computer*, vol. 35(2), Feb. 2002.

[16] N. L. Binkert, R. G. Dreslinski, J. Eskilson, and D. Forsgren, "The m5 simulator: Modeling networked systems," in *IEEE Micro*, vol. 26, no. 4, July/August, 2006.

[17] C.-K. Luk, R. Cohn, R. Muth, and H. Patil, "Pin: Building customized programanalysis tools with dynamic instrumentation," in *Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation, ACM New York, NY, USA*, pp. 190–200, 2005.

[18] A. Srivastava and A. Eustace, "Atom: A system for building customized program analysis tools," in *Proceedings of the ACM SIGPLAN 1994 conference on Programming language design and implementation. ACM New York, NY, USA*, 1994.

[19] W. Wang, L. Qiao, G. Yang, and Z. Tang, "Performance analysis of the 2-d networks-on-chip," *Journal of Computer Research and Development*, vol. 46(10), 2009.

[20] F. Dahlgren and P. Stenstrom, "Evaluation of hardware-based stride and sequential prefetching in shared-memory multiprocessors," *IEEE Transactions on Parallel and Distributed Systems*, vol. v.7 n.4, 1996.

[21] K. J. Nesbit and J. E. Smith, "Data cache prefetching using a global history buffer," in *International Symposium on High-Performance Computer Architecture, Madrid, Spain*, Feb. 2004.

[22] K. J. Nesbit, A. S. Dhodapkar, and J. E. Smith, "Ac/dc: An adaptive data cache prefetcher," in *Proceedings of the 13th International Conference on Parallel Architecture and Compilation Techniques*, 2004.

[23] D. Joseph and D. Grunwald, "Prefetching using markov predictors," in *ISCA '97: Proceedings of the 24th annual international symposium on Computer architecture*, 1997.

[24] F. Dahlgren and P. Stenstrom, "Effectiveness of hardware-based stride and sequential prefetching in shared-memory multiprocessors," in *HPCA '95: Proceedings of the 1st IEEE Symposium on High-Performance Computer Architecture*, 1995.

[25] Z. Hu, M. Martonosi, and S. Kaxiras, "Tcp: Tag correlating prefetchers," in *HPCA '03: Proceedings of the 9th International Symposium on High-Performance Computer Architecture*, 2003.

[26] P. Sweazey and A. J. Smith, "A class of compatible cache consistency protocols and their support by the ieee futurebus," in *Proceedings of the 13th annual international symposium on Computer architecture*, 1986.