The Study of Statistical Simulation for Multicore Processor Architectures

Jongbok Lee

Dept. of Information and Communications Engineering
Hansung University
Seoul, Republic of Korea
Email: jblee@hansung.ac.kr

***Abstract*—The execution-driven or trace-driven simulation is often used for the performance analysis of widely used multicore processors in the initial design stage. However much time and disk space is necessary. In this paper, statistical simulations are performed for high performance multicore processors with various hardware configurations. For the experiment, the SPEC2000 benchmarks programs are used for the statistical profiling and synthesis. As a result, the performance obtained by the statistical simulation is comparable to that of the trace-driven simulation, with a tremendous reduction in the simulation time.**

Keywords–multicore processor, statistical simulation

I. INTRODUCTION

Currently, multicore processors are widely used for enhancing the performance of the computer system, such as smart phones, tablet PCs, notebook computers, desk top computers, etc. [1][2][3]. Since extensive simulations are necessary in the initial design stage of these multicore processors, the execution-driven simulation or trace-driven simulation is generally used. The execution-driven simulation is accurate but requires excessive time, whereas the trace-driven simulation is less accurate with the benefit of relatively reduced time. In addition, the trace-driven simulation has the disadvantage of requiring excessive disk space.

In order to address these obstacles, various alternative techniques have been studied. In the statistical simulation, the statistical characteristics of the processor architecture and the benchmark programs are collected. The statistical profiling is the collection of the characteristic distribution of the programs and the processor architectures. And then, new instruction traces are synthesized upon these statistical profiles. Since the new instruction traces are randomly generated by the statistical profiles, they represent the characteristics of each benchmark implicitly. Finally, the statistical simulation is performed with the new instruction traces on a simple statistical trace-driven simulator, which drastically reduces the simulation time [4][5][6][7][8]. Therefore, the statistical simulation can be useful for measuring the performance of multicore processors in the initial design stage, with the reduced time and space.

In this paper, the SPEC2000 integer benchmark programs are used for estimating the performance of multicore processors using statistical simulation. The result is compared with the performance of the trace-driven simulation by calculating the relative errors. This paper is organized as follows. In the Section 2, the statistical profiling will be discussed. The simulation environment will be described in Section 3. In Section 4, the simulation results will be analyzed. Finally, Section 5 concludes our paper.

II. THE STATISTICAL PROFILING

The statistical simulation consists of four stages, as shown in the Figure 1.

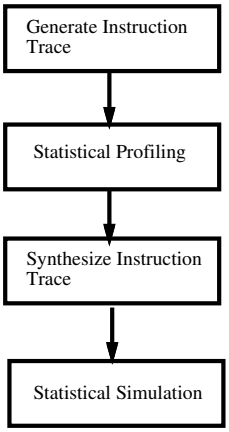


Figure 1. The statistical simulation process

The first stage is the generation of the instruction traces, which is the same as the conventional method. Each benchmark program is executed and its instruction traces are obtained by the instruction trace generator.

Secondly, the statistical profiling and the statistical data collection are performed. In this process, the instruction is analyzed and the intrinsic characteristic and the local property of each benchmark program is collected. The intrinsic characteristic of each benchmark program consists of instruction class distribution, the number of operand register distribution, and the data dependency among instructions. The instruction class distribution is obtained by reducing the original instruction set to that of only nine simple instructions. The register dependency among instructions is defined as the distance between the preceding instruction that includes the destination register and the subsequent instruction that has the source register on which it depends. These characteristics of benchmark are independent of the given microprocessor architecture, but dependent only on the instruction set and the compiler. The local characteristics include task misprediction rates, and

 ISBN: 978-1-61208-394-0

various cache miss rates, etc. [9]. These are affected by the microprocessor hardware. This procedure is performed only once during the whole process.

Thirdly, using the collected statistical profiles of the second stage, a new benchmark is synthesized by the random number generation. A random number between 0 and 1 is generated and mapped onto the cumulative distribution function obtained from statistical profile in order to synthesize a new instruction trace.

Finally, the synthesized instruction trace is input to the statistical multicore simulator with the task prediction hit rates and the cache hit rates. Once all the data from the statistical profile and the synthetic instruction traces are secured,various experiments can be conducted by modifying the hardware architecture such as the number of cores, the task size, instruction fetch rates, the number of pipeline stages. Thus, a number of various simulations can be performed in a dramatically reduced time and space.

III. The Simulation Environment

A. The multicore processor architecture

Figure 2 shows the multicore processor with N cores. Each core is an in-order or out-of-order superscalar processor which can execute instructions in a task [2][9].

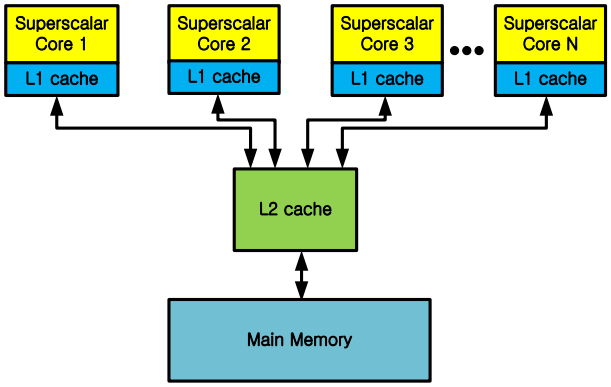


Figure 2. The multicore processor

In addition, it has a L1 instruction cache and a L1 data cache. For the cache coherency of the L1 data cache, MESI (Modified Exclusive Shared Invalidate) protocol is utilized [10]. If the data in the L1 data cache associated with a core is over-written by another core, it is invalidated. The L2 cache is shared among the cores, which is connected with the main memory.

The superscalar processor core is allocated with a task which consists of a number of instructions. The fetched instructions in the task are decoded, renamed, executed, and written back. When all the instructions in the task are retired and becomes empty, new instructions of task are fetched. If the task is mispredicted, the fetch is aborted, and all the remained instructions in the task are squashed. Since the instructions are renamed, the instructions can be issued and executed out-of-order as long as there is no true-dependency. Although the instructions can be retired out-of-order, the instructions are inserted into the reorder buffer and committed in-order as to preserve the original program order.

The detailed architecture configurations and cache parameters for each core are listed in Table I. The number of simulated cores are 2, 4, and 8. Each core is assigned one task respectively. Since the small task size cannot take the benefit of the instruction level parallelism, the task sizes are set to 4, 8, and 16. The functional unit of each core consists of a number of ALUs (Arithmetic Logic Units), load/store units according to each configuration.

TABLE I. ARCHITECTURE CONFIGURATION FOR EACH CORE.

Item		Value
number of cores		2,4,8
number of task		1
task size		4,8,16
fetch rate		2,4,8
issue rate		2,4,8
retire rate		2,4,8
functional unit	integer ALU	2,4,8
	load/store	1,2,4
L1-instruction cache		64 KB, 2-way set assoc., 16 B block, 10 cycles miss penalty
L1-data cache		64 KB, 2-way set assoc., 32 B block 10 cycles miss penalty
task address cache		2K entry
task predictor		14-bit global history based 6 cycle mispred. penalty

For the memory disambiguation, load-store and store-store pairs are inhibited from the speculative execution when the effective addresses are matched, within or among the cores. The L1 instruction cache and L1 data cache for each core is 64 KB, and it is designed as 2-way set associative. This is because the data cache hit ratio can be degraded by using MESI protocol among multicores. Tasks are predicted using the Two-level Adaptive Task Prediction scheme, and the task address cache has the size of 2048 entries. Since we do not model the main memory, the hit ratio of L2 cache is assumed to be 100 %.

B. The multicore processor simulator

Figure 3 depicts how the developed simulator works. *Initialize* function initializes all the associated variables, and *Grouping*, *Create_Window*, and *Fetch_One_Instr* function fetches new instructions every cycle. The instruction fetched by *Get_Node* function is renamed at *Rename* function by receiving timestamps. After the instruction is renamed, it is inserted into the instruction window by *Insert* function. At the *Issue* function, the instruction in the window can be retired as long as the corresponding functional unit is available and its time stamp is less than or equal to the current cycle. For the multicore simulation, *Grouping* function fills an instruction into n-cores, and *Issue* function deletes instructions according to their timestamps. This process is repeated until all the fetched instructions are deleted so that all the instruction windows are empty. Then, the cores are filled again with instructions with *Grouping* function. Since the cycle is incremented for each process, the core which spends the longest cycles determines the global cycle. If the total number of executed instructions is divided by the number of cycles spent, then IPC (Instruction per Cycle) can be obtained.

The seven SPEC 2000 integer benchmark programs that are used for the input are *bzip2,crafty, gap, gcc, gzip, parser*,

 ISBN: 978-1-61208-394-0

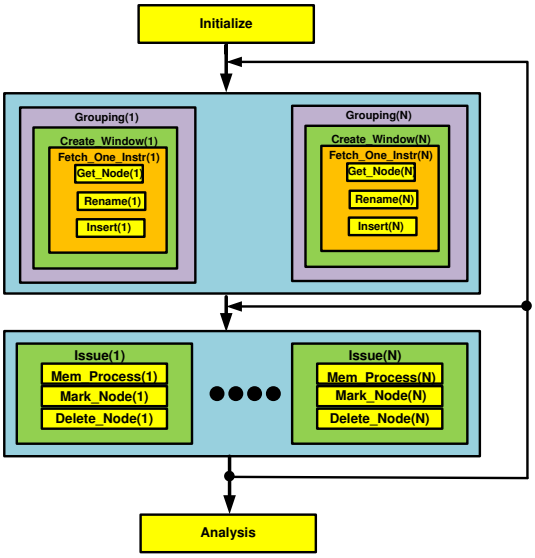


Figure 3. The flow chart of the multicore processor simulator

and *twolf*, as shown in Table II. The programs are compiled by SimpleScalar cross C compiler to obtain executables under Linux 3.3.4 [11]. The execution files are run with SimpleScalar to obtain 100 million MIPS IV instruction traces. While these are used as input for the multicore processors, the task-level parallelism is mapped onto each core.

TABLE II. SPEC 2000 BENCHMARK PROGRAMS

benchmark	description
bzip2	compression
crafty	chess game
gap	group theory interpreter
gcc	C programming language compiler
gzip	compression
parser	word processor
twolf	placement and global routing

IV. The Simulation Results

Figure 4 presents the simulation results of running SPEC 2000 integer programs on the three different task sizes of dual core, quad core, and octa core processors. The performance results obtained by the general trace-driven simulation and the statistical simulation are compared in parallel. Figure 4a and 4b are the result of the multicore processors with the maximum task size of four. For the dual core processors, the trace-driven simulation brings the geometrical mean of 1.1 IPC, whereas the statistical simulation results in 1.3 IPC. For the quad core processors, the trace-driven simulation and the statistical simulation results in 2.0 IPC and 2.4 IPC, respectively. Finally, the respective performances for the octa core processors are 3.2 IPC and 4.3 IPC. Unlike the respective relative error of 18 % and 20 % of dual core and quad core processors, the octa core processors scores the relative error of 25 % in the average.

Figure 4c and 4d show the results with the maximum task size of eight. The performance of the dual core, the quad core, and the octa core processors measured by the trace-driven simulations are 1.3 IPC, 2.4 IPC, and 4.0 IPC, respectively. The corresponding values by the statistical simulations are 1.8 IPC, 3.1 IPC, and 5.4 IPC. The average relative error results in 35 %.

Finally, Figure 4e and 4f presents the comparison result when the maximum task size is sixteen. For the dual core processors, the trace-driven simulation brings 1.5 IPC, whereas the statistical simulation results in 2.1 IPC. For the quad cores, the respective values are 2.8 IPC and 3.7 IPC. And for the octa cores, the relative error is increased by the results of 4.7 IPC and 6.4 IPC, respectively. However, the average relative error does not exceed 35 %.

As the result shows, the performance of multicore processors evaluated by statistical simulation has the similar tendency of the trace-driven simulation with the average relative error of 18 % to 35 %. Encouragingly, the average statistical simulation time is 9 seconds per benchmark program, which is 30 times faster than the trace-driven simulation. Therefore, this compensates for not being highly accurate.

V. Conclusions

In this paper, the performance of multicore processors has been evaluated and analyzed by the statistical simulation. Since the statistical simulation takes advantage of the newly synthesized instruction traces by statistical profile, the disk space is saved and the average simulation time is drastically reduced. Although the experiment shows the average relative error of 18 % to 35 %, the simulation time has been drastically reduced to 1/30.

For future research, we will study the method to further improve the accuracy of the statistical simulation, as well as expanding our scope to the multicore embedded and multicore digital signal processor architectures.

Acknowledgement

The author would like to thank Hansung University for the financial support of this research.

References

[1] T. Ungerer, B. Robic, and J. Silk, “Multithreaded Processors,” The Computer Journal, vol. 45, no. 3, 2002.

[2] S. W. Keckler, K. Olukotun, and H. P. Hofsee, Multicore Processors and Systems. Springer, 2009.

[3] M. Monchiero, “How to simulate 1000 cores,” ACM SIGARCH Computer Architecture News archive, vol. 37, no. 2, May 2009, pp. 10–19.

[4] D. B. Noonburg and J. P. Shen, “A Framework for Statistical Modeling of Superscalar Processor Performance,” in Proceedings on Third International Symposium on High Performance Computer Architecture, 1997.

[5] R. Carl and J. E. Smith, “Modeling Superscalar Processors via Statistical Simulation,” in Workshop on Performance Analysis and Its Impact on Design, Jun. 1998.

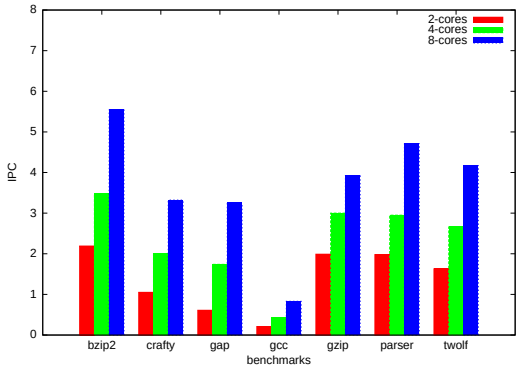
[6] L. Eeckout, K. D. Bosschere, and H. Neefs, “Performance Analysis through Synthetic Trace Generation,” in International Symposium on Performance Analysis of Systems and Software, Apr. 2000.

[7] D. Genbrugge and L. Eckhout, “Chip multiprocessor design space exploration through statistical simulation,” IEEE Transactions on Computers, vol. 58, no. 12, Dec. 2009, pp. 1668–1681.

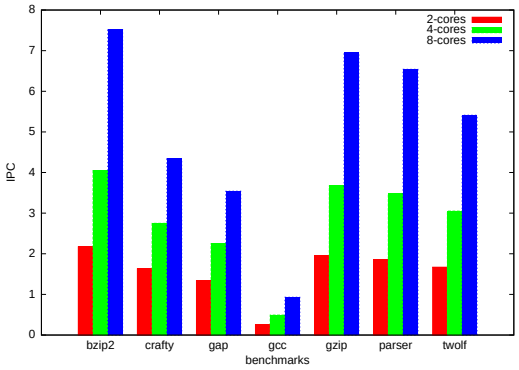
[8] A.Rico, A. Duran, F. Cabarcas, A. Ramirex, and M. Valero, “Trace-driven simulation of multithreaded applications,” in ISPASS, 2011.

[9] T. N. Vijaykumar and G. S. Sohi, “Task selection for a multiscalar processor,” in 31st International Symposium on Microarchitecture, Dec 1998.

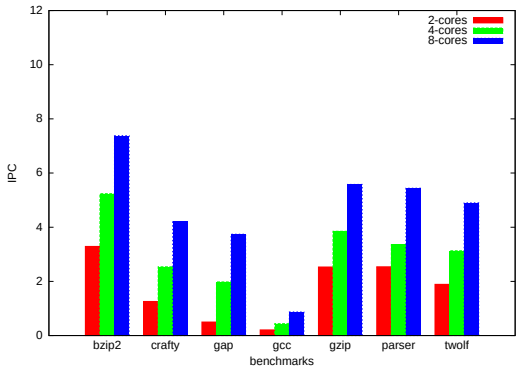
 ISBN: 978-1-61208-394-0



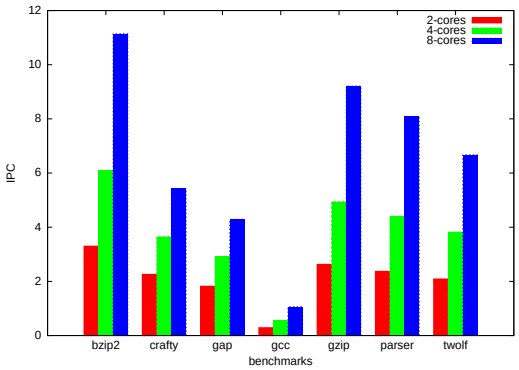
(a) simulated, task size =4



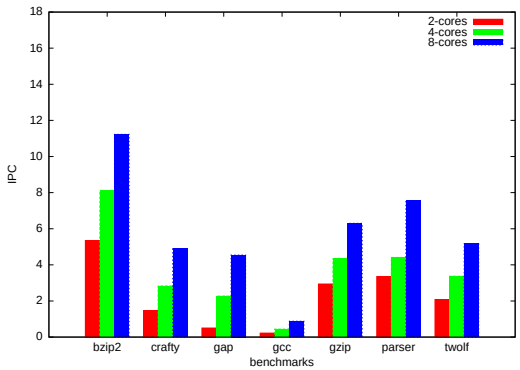
(b) statistical, task size =4



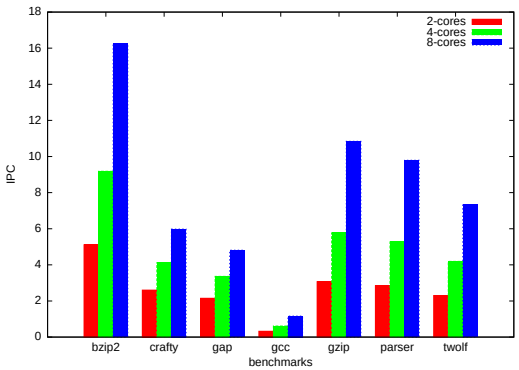
(c) simulated, task size =8



(d) statistical, task size =8



(e) simulated, task size =16



(f) statistical, task size =16

Figure 4. Performance results of the trace-driven and the statistical simulation

[10] M. S. Papamarcos and J. H. Patel, “A low-overhead coherence solution for multiprocessors with private cache memories,” in Proceedings of the 11th Annual International Symposium on Computer Architecture, Jun 1984, pp. 348–354.

[11] T. Austin, E. Larson, and D. Ernest, “SimpleScalar : An Infrastructure for Computer System Modeling,” Computer, vol. 35, no. 2, Feb. 2002, pp. 59–67.

 ISBN: 978-1-61208-394-0