Performance Evaluation of SMT and CMP Processor Using Super Scalar Simulator

Komala.S[1], Ravikumar.G[2]

[1]Student, Computer Science Department
[2]Assistant Professor, Computer Science Department

Abstract- Multithreading and prefetching are the techniques used to increase the performance of the processor. Multithreading executes another concurrent thread when running thread encounters a catch miss. Prefetching increases the single thread performance by predicting the data address in advance. This paper presents the performance of standard benchmarks in simultaneous multithreading processor (SMT) and chip multiprocessor (CMP). SMT is a tightly coupled system where the processor and the cache are dependent on each other. CMP is a loosely coupled system with independent processor and cache. The processor is demonstrated by cyclic accurate execution driven simulator and the processor performance are evaluated based on execution speed and execution time.

Index Terms- SMT, CMP, Parallel Computing, Distributed Computing.

I. INTRODUCTION

As the speed gap between processor and memory system increases, a processor spends more time on memory stalls. To tolerate large memory latency, there have been a number of proposals for data prefetching [1]. Recently, a novel thread based prefetching technique called pre execution has received much attention in the research community.

Simultaneous multithreading (SMT) process program with multiple processors [6]. These processors share their common memory location. SMT processor balances the workload dynamically among the processors [2]. SMT processor resources are duplicated to handle more than one computational entity. Also there are large numbers of dependencies between instructions and multiple entities are executed to occupy unused resources.

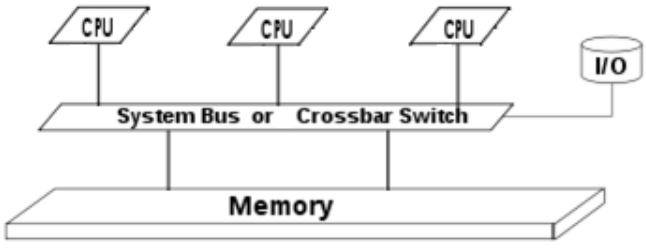

Fig. 1: The SMT Architecture.

The next performance increasing technique is chip multiprocessor (CMP). CMP duplicates the entire subsystem on a single die [7]. Each core will have their primary and secondary cache. It is also possible to modify the processor with additional logic to behave as a dual core die.

These two processors are evaluated based on the performance of multi programmed environment [5]. Both the processors have limitations related to resources shared by multiple thread contents and are centered on data and instruction path connecting one processor with another.

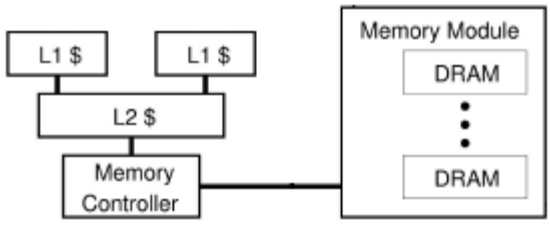

Fig. 2: The CMP Architecture.

This paper is based on performance of simultaneous multithreading processor (SMT) and chip multiprocessor (CMP). SMT is a tightly coupled system where the processor and the cache are dependent on each other. CMP is a loosely coupled system with independent processor and cache. The processor is demonstrated by cyclic accurate execution driven simulator and the outputs are evaluated.

II. RELATED LITERATURE

Pre-execution technique can be applied based on hardware and software configurations. Pre-execution techniques [7] for simultaneous multithreading (SMT) processor have been proposed and the performances are obtained by executing in simple scalar simulator. Optimizations can be made statically and dynamically to improve processor performance. Source-to-source compilation of pre-execution thread with program slicing, prefetch conversion and thread scheme selection algorithms are proposed [4].

Instruction traces are analyzed by speculative per-computation [1] and data-drive multithreading [4] to pre-execute cache miss loads. Another technique used for pre-execution is execution-based prediction [8], which extracts the pre-execution code from binaries without instruction traces.

Dynamic speculative pre-computation [10] and dependence graph pre-computation [9] perform pre-execution which extract the code using trace processing hardware. Slip stream processor [15] identifies fault tolerance using speculative compute engine which runs on SMT and CMP processor. CMP processors

resources and caches are independent of each other so, whenever a change is made that does not reflect the entire processor system.

III. IMPLEMENTATION DETAILS

3.1 SESC Utilities

The performance of SMT and CMP processor is evaluated by super scalar simulator (SESC) [11]. Before building the super scalar source code it is necessary to build utility files. Super scalar utility is a group of building programs to run SESC. Super scalar utility is a cross-compiler tool chain which is used to compile programs on PC for running on MIPS. Before building utility file the compiler version is to be changed. SESC is built based on gcc 3.4 compiler version. Newer version of gcc is not compactable with older one.

To change the compiler version, files from deb packages are downloaded and installed in the system. The gcc versions are listed down and the needed version is selected based on the option. With utility tar file, original utility files are extracted and moved to the home directory. Build common file configuration is changed which is present in build-mipseb-linux. All the utility files are executed by installing bison, flex and lib.

3.2 SESC Environment

To build super scalar environment in the system, download the source code from cvs and place it in the home directory. Create a build folder and configure the super scalar simulator by installing bin utility which is necessary for building program running on super scalar simulator. If errors persist include limits and stdint header files. Install dev files and run make command. Successfully simulator is installed and test cases are checked.

3.3 Benchmarks

To test this experiment, benchmarks of SPEC CPU2000 and SPLASH2 are evaluated using super scalar (SESC) simulator. These benchmarks are given by standard performance evaluation corporation (SPEC) to identify and improve the processor performance. The simulation results are obtained by executing the benchmarks in SMT and CMP processor.

3.4 SMT Processor

There are number of configurations available within the super scalar simulator. To execute the benchmarks in SMT processor, enable the configuration file of simultaneous multithreading processor and create an environment by adopting the changes in library files. The benchmarks are executed and the results are tabulated in TABLE 1

TABLE 1
Applications Executed in SMP Processor

Applications	Execution Speed (KIPS)	Execution Time (sec)	Simulation Time (msec)
Crafty	949.830	21.750	7.787
Fft	958.939	0.330	0.078
Radix	923.964	41.640	32.809
Barnes	206.00	0.010	0.012
Smatrix	1073.764	2.080	0.383
Mcf	211.00	0.010	0.012

3.5 CMP Processor

To execute the benchmarks in CMP processor some changes are made by moving the configuration file to the executing environment. This change enables the chip multiprocessor and the benchmarks are executed. The results are tabulated in TABLE 2.

TABLE 2
Applications Executed in CMP Processor

Applications	Execution Speed (KIPS)	Execution Time (sec)	Simulation Time (msec)
Crafty	418.267	49.390	4.517
Fft	351.611	0.900	0.082
Radix	293.156	131.240	29.800
Barnes	206.00	0.010	0.014
Smatrix	271.377	8.230	0.33
Mcf	105.500	0.020	0.015

IV. CONCLUSION

The performance of SMT and CMP processor using super scalar simulator is presented in this paper. The future work will concentrate on improving the performance of processors by creating helper thread to reduce cache miss rate.

REFERENCES

[1] A. Roth and G. Sohi, “Speculative Data-Driven Multithreading,” Proc. Seventh Int’l Symp. High-Performance Computer Architecture (HPCA ’01), pp. 37-48, Jan. 2001.

[2] C.-K. Luk, “Tolerating Memory Latency through Software-Controlled Pre-Execution in Simultaneous Multithreading Processors,” Proc. 29th Int’l Symp. Computer Architecture (ISCA ’01), June 2001.

[3] [C.B. Zilles and G.S. Sohi, “Understanding the Backward Slices of Performance Degrading Instructions,” Proc. 27th Int’l Symp.Computer Architecture (ISCA ’00), pp. 172-181, June 2000.

[4] D. Kim and D. Yeung, “Design and Evaluation of Compiler Algorithms for Pre-Execution,” Proc. 10th Int’l Conf. ArchitecturalSupport for Programming Languages and Operating Systems (ASPLOS ’02), pp. 159-170, Oct. 2002.

[5] D. Kim, S.-W. Liao, P.H. Wang, J. del Cuvillo, X. Tian, X. Zou, H. Wang, D. Yeung, M. Girkar, and J.P. Shen, “Physical Experimentation with Prefetching Helper Threads on Intel’s Hyper-Threaded Processor,” Proc. Second Int’l Symp. Code Generation and Optimization (CGO ’04), pp. 27-38, Mar. 2004.

[6] G.K. Dorai and D. Yeung, “Transparent Threads: ResourceAllocation in SMT Processors for High Single-Thread Performance,” Proc. 11th Ann. Int’l Conf. Parallel Architectures and Compilation Techniques (PACT ’02), Sept. 2002.

[7] J.A. Brown, H. Wang, G. Chrysos, P.H. Wang, and J.P. Shen, “Speculative Precomputation on Chip Multiprocessors,” Proc. Sixth Workshop Multithreaded Execution, Architecture and Compilation (MTEAC ’02), Nov. 2002.

[8] J.D. Collins, D.M. Tullsen, H. Wang, and J.P. Shen, “Dynamic Speculative Precomputation,” Proc. 34th Int’l Symp. Microarchitecture (MICRO ’01), Dec. 2001.

[9] J.D. Collins, H. Wang, D.M. Tullsen, C.J. Hughes, Y. fong Lee, D. Lavery, and J.P. Shen, “Speculative Precomputation: Long-Range Prefetching of Delinquent Loads,” Proc. 28th Int’l Symp. Computer Architecture (ISCA ’01), pp. 14-25, July 2001.

[10] J.Lee, Y.Solihin, and J.Torrellas, "Automatically Mapping Code in an Intelligent Memory Architecture," Proc. Seventh Int'l Symp. High-Performance Computer Architecture (HPCA '01), Jan. 2001.

[11] J.Renauetal.,SESC, http://sesc.sourceforge.net, 2004.

[12] M. Dubois and Y.H. Song, "Assisted Execution," Technical Report 98-25, Dept. Electrical Eng.—Systems, Univ. of SouthernCalifornia, Oct. 1998.

[13] M. Weiser, "Program Slicing," Proc. Fifth Int'l Conf. Software Eng. (ICSE '81), pp. 439-449, 1981.

[14] R.Cooksey, D.Colarelli, and D. Grunwald, "Content-Based Prefetching: Initial Results," Proc. Second Workshop IntelligentMemory Systems, pp. 33-55, Nov. 2000.

[15] W. Zhang, B. Calder, and D. Tullsen, "Dynamic Speculative Precomputation," Proc. Fourth IEEE Int'l Symp. Code Generation and Optimization (CGO '06), Mar. 2006.

[16] Y. Solihin, J. Lee, and J. Torrellas, "Using a User-Level Memory Thread for Correlation Prefetching," Proc. 29th Int'l Symp.Computer Architecture (ISCA '02), May 2002.

[17] Yan Solihin_, Jaejin Lee_, and JosepTorrellas, "Prefetching in an Intelligent Memory Architecture Using a Helper Thread," Proc. 28th Int'l Symp. Computer Architecture, May 2001.

AUTHORS

First Author –Komala.S, Student, Computer Science Department, Email: koms10cse@gmail.com
Second Author – Ravikumar.G, Assistant Professor, Computer Science Department, Email: rgrtvr2007@yahoo.co.in