An Optimal Simultaneous Multi-Threading Architecture with In-order Issue and Completion

In-Pyo Hong, Byung-In Moon, Moon-Gyung Kim, Jae-Eok Kim, Yong-Surk Lee
Processor Laboratory, Department of Electric and Electronic Engineering, Yonsei University, Seoul, Korea
Homepage: http://mpu.yonsei.ac.kr/
E-mail: necross@dubiki.yonsei.ac.kr

Abstract

This paper proposes an optimal ARM-compatible SMT (Simultaneous Multi-Threading) architecture with in-order issue and completion. SMT architecture uses TLP (Thread Level Parallelism), such that issue slots can be filled with instructions from multiple independent threads without complicated out-of-order issue and completion schemes. Thus, the in-order issue and completion technique is utilized to simplify hardware design and reduce area. This architecture is implemented on a cycle-based/execution driven simulator and optimized by using various architecture options to maximize throughput and decrease hardware complexity. The simulation results show that a well-tuned in-order SMT architecture achieves triple the IPC (Instructions Per Cycle) when compared to conventional superscalar architecture.

I. Introduction

Currently, most high performance processor architectures are based on the superscalar RISC technology. Superscalar processors search multiple independent instructions from a single instruction stream and execute them simultaneously. However, because most instructions from a single thread are dependent on one another, ILP (Instruction Level Parallelism) is seriously limited. Thus, the maximum throughput of 8-way issue superscalar processors is about 2 IPC.[1]

To overcome this architectural limit, SMT [2] architecture is proposed. SMT improves the throughput of superscalar architecture by utilizing TLP. In an SMT processor, because there is no dependency among instructions from independent threads, more instructions can be issued simultaneously to functional units than in a superscalar processor.

In this paper, an SMT architecture with a simple in-order issue and completion scheme is proposed and optimized using an architecture simulator. Chapter 2 presents our SMT ARM architecture. In Chapters 3 and 4, the simulation methodology and results are described, respectively, and in Chapter 5, we summarize our simulation results and propose an optimal SMT architecture.

II. SMT ARM Architecture

The proposed SMT architecture is compatible with ARM™ [3] instruction set architecture (ISA)

version 5. Because ARM architecture targets low performance embedded processors, many features of ARM™ ISA, such as conditional executions, branch-like instructions, and multi-cycle instructions, are not suitable for superscalar or SMT architecture. These features cause the pipeline control and dependency checks to become complex, but this situation can be overcome by adopting a simple in-order issue and completion technique.

2.1 Architecture overview

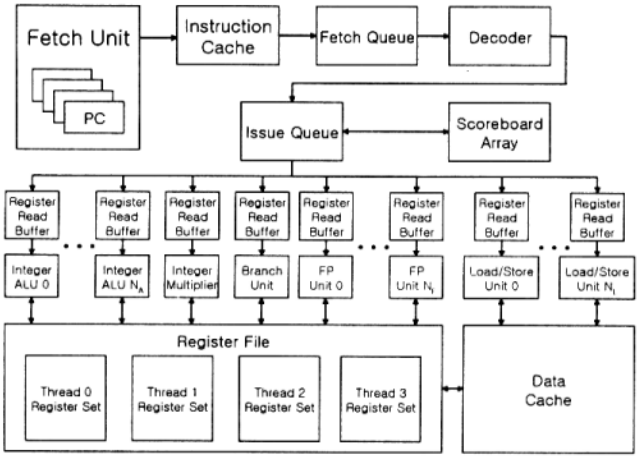

Fig 1. Overall architecture of SMT ARM

Fig. 1 shows an overall block diagram of SMT ARM architecture which supports four active threads. In comparison with conventional superscalar architectures, a few changes are required. There are multiple register sets, multiple PCs (Program Counters), and multiple scoreboard[4] arrays which correspond to multiple active hardware threads. Except for these features, almost all of the hardware resources are shared by all active threads dynamically. In order to classify instructions according to their owner threads, the thread IDs are attached to each fetched instruction and flow through pipeline stages together with those instructions.

2.2 Pipeline structure of SMT ARM

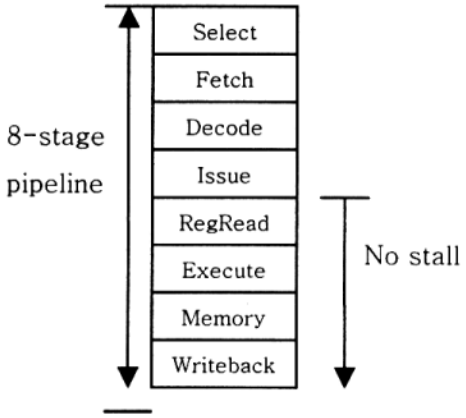

Fig 2. Pipeline of SMT ARM architecture

Fig. 2 shows the pipeline structure of the SMT ARM architecture. The Select and RegRead stages are added to support SMT.

In SMT architecture, due to the multiple hardware threads, the fetch unit must select the appropriate number of threads to fetch in the next cycle. The fetch thread selector calculates the priorities of the threads by using a pre-defined priority algorithm, and allocates instruction cache ports according to their priority order. This fetch thread selection occurs in the Select stage. The thread selection priority algorithms are as follows:

- ICOUNT_IFQ

  Give the highest priority to the thread that has the smallest number of instructions in the IFQ (Instruction Fetch Queue).

- ICOUNT_Q

  Give the highest priority to the thread that has the smallest number of instructions in the IFQ and IIQ (Instruction Issue Queue).

- ICOUNT_ALL

  Give the highest priority to the thread that has the smallest number of instructions in the IFQ, IIQ and all pipeline stages.

- ICOUNT_BHB

  Give the lowest priority to the thread that has the largest number of instructions in the BHB (Branch History Buffer).

- ICOUNT_LB

  Give the lowest priority to the thread that has the largest number of instructions in the LB (Load Buffer).

- IQOL

  Give the lowest priority to the thread that has an instruction closest to the top of the IIQ.

- RR

  Round-Robin

In the Fetch stage, the fetch unit accesses instruction cache ports which are allocated in the previous cycle and stores fetched instructions into the IFQ. Branch prediction using BTB is done in the Fetch stage simultaneously. Then, instructions in the IFQ are decoded in order and the decoded information is stored in the IIQ.

The issue logic performs dependency and condition checks by accessing scoreboard arrays and status registers, and issues instructions in order within a thread. However, the issue logic does not check dependency between instructions issued from different threads, and the IIQ order is ignored. That is, the issue strategy of the SMT ARM is in-order within a thread, and out-of-order among threads. Thus, since empty entries that will lower issue-performance may exist between valid entries on the IIQ, compaction on the IIQ is required. When the number of dependency-free instructions is larger than the maximum issue bandwidth, issue priority algorithms are required. Those algorithms are as follows:

- OLDEST

  Give the highest priority to the oldest instruction in the IIQ.

- ICNT_FU

  Give the highest priority to the thread that has the smallest number of instructions in functional units.

- ICNT_MISS

  Give the highest priority to the thread that has the smallest number of cache miss counts.

Instructions issued successfully do not suffer any pipeline stall in the following pipeline stages: RegRead, Execute, Memory, and Writeback. Instead of stalling, branch instructions that produce branch prediction miss and instructions that generate exception are invalidated. In the RegRead stage, functional units obtain source operands from register file or forwarding path, and activate the corresponding state machine for multi-cycle instructions if necessary. The Execute, Memory and Writeback stages are the same as those of common superscalar architectures.

III. Methodology

The proposed SMT ARM architecture is implemented as a cycle-based/execution-driven simulator written in C-language. Because the simulator has many architectural options, the performance of various architectural configurations can therefore be examined.

Table 1. Workloads for performance evaluation

Benchmark Suite	Program
SPEC CPU 2000	parser
	twolf
	vortex
	mcf
ADS example	Sorts
	Dhrystone

Workloads are generated by using the compiler included in ADS (ARM Development Suite). [5] The simulator loads and executes multiple independent workloads as threads. Workloads used in this research are shown in table 1.

VI. Performance Evaluation Results

Important factors affecting the overall performance of the SMT architecture are the the fetch priority algorithm, issue priority algorithm, cache/BTB performance, fetch/issue bandwidth. Table 2 shows the performance of the fetch priority algorithms. In this simulation, the number of threads is eight, and the issue priority algorithm is fixed to OLDEST. All algorithms except ICOUNT_BHB and ICOUNT_LB show satisfying results above four instructions per cycle. Because ICOUNT_BHB and ICOUNT_LB have static characteristics, which mean that priority among threads changes very slowly, the fetch unit successively fetches instructions from the high-priority fixed thread. Therefore, the SMT core cannot fully use TLP among threads. As a result, RR is the simplest algorithm to implement in hardware, and it shows moderate throughput.

Table 2. Performance of fetch priority algorithm (IPC)

Algorithm	Fetch rate	Execution rate
ICOUNT_IFQ	6.2318	4.6911
ICOUNT_Q	6.3786	4.9661
ICOUNT_ALL	6.7201	4.9783
ICOUNT_BHB	5.5374	2.8848
ICOUNT_LB	5.9460	3.0740
IQOL	6.5601	4.7194
RR	6.2945	4.6680

Many factors, such as dependencies, condition checks, and the availability of functional units, affect the issue operation. Therefore, the choice of an issue priority algorithm has less impact on performance. Table 3 shows the performance of three issue priority algorithms.

Table 3. Performance of issue priority algorithm (IPC)

Algorithm	Issue rate	Execution rate
OLDEST	5.3211	4.6680
ICNT_FU	5.2031	4.6174
ICNT_MISS	5.2185	4.5083

After analyzing the simulation results, we found that in order to optimize the cost/performance ratio, the fetch bandwidth, issue bandwidth and number of threads must be the same value. Therefore, in the following results, the numbers of threads, fetch bandwidth and issue bandwidth have the same value. Also, because the SMT architecture executes multiple threads simultaneously, the thrashing effect may occur on the cache. Thus, the instruction and data

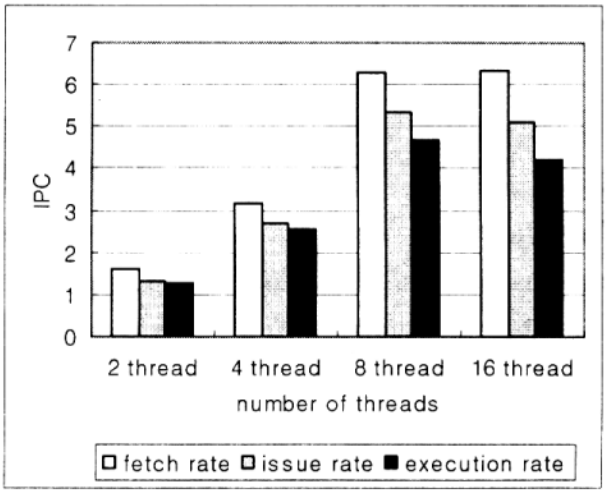

Fig 3. Throughput of SMT ARM architecture as a function of the number of threads

cache of the SMT ARM is 8-way set associative and 64KB respectively. Fig 3 shows the overall performance of the SMT ARM architecture. The throughput of the optimal 8-thread configuration is 4.7 IPC, which is almost the triple the throughput of conventional high-performance superscalar architecture.

Table 4. Cache miss and BTB replacement ratio as a function of the number of threads (%)

number of threads		2	4	8	16
Cache miss	I-cache	0.01	0.02	0.06	2.16
	D-cache	2.51	5.44	6.19	12.45
BTB replacement		0.32	4.28	20.67	34.83

As the number of supported threads increases, the throughput improves. On the other hand, supporting a large number of threads can cause the thrashing effect on the cache and BTB. Table 4 shows that the 16-thread configuration has the largest cache miss ratio and BTB replacement count. As a result, although higher TLP can be found on a 16-thread configuration, the throughput is less than that of an 8-thread configuration.

V. Conclusion

In this paper, an SMT architecture with in-order issue and completion is proposed and optimized. Many fetch and issue priority algorithms are evaluated and its performances are presented. Since the current cache/BTB performances are limited, an SMT architecture which supports 8 or less threads are reasonable to implement. Even though in-order issue and completion technique is adopted, the properly-tuned 8-thread SMT architecture is able to execute an average of 4 ~ 5 instructions per cycle, whereas such high throughput cannot be achieved with conventional high-performance superscalar architecture.

Acknowledgement

This research is supported by Samsung Electronics and the NRL (National Research Laboratory) project of the Ministry of Science and Technology, Republic of Korea.

Reference

[1] D.M. Tullsen, S.J. Eggers, J.S. Emer, H.M. Levy, J.L. Lo, and R.L. Stamm, "Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor," Proc. 23rd Annual International Symposium on Computer Architecture, pp.191-202, May 1996.

[2] Susan J. Eggers, Joel S. Emer, Henry M. Levy, Jack L. Lo, Rebecca L. Stamm, Dean M. Tullsen, "SIMULTANEOUS MULTITHREADING: A Platform for Next-Generation Processors", September/October 1997 IEEE Micro, p.p. 12~19

[3] ARM, *ARM Architecture Reference Manual, Part A CPU Architecture*, 1996

[4] Mike Johnson, "*Superscalar Microprocessor Design*", Prentice-Hall, pp. 107~126, 1991

[5] ARM, *ARM Developer Suit version 1.1, Compiler, Linker and Utilities Guide*, 1999