International Journal of Modelling and Simulation, Vol. 18, No. 3, 1998, pp.201–207

A MODEL FOR PREDICTING UTILIZATION OF MULTIPLE PIPELINES IN MTMP ARCHITECTURE

Yamin Li and Wanming Chu

Computer Architecture Laboratory
Department of Computer Hardware
The University of Aizu
Aizu-Wakamatsu, 965-80 Japan
Email: {yamin,w-chu}@u-aizu.ac.jp

Abstract

The conventional single-threaded multiple-pipelined processor is not capable of using multiple pipelines efficiently, and so the processor performance suffers. This paper investigates a multiple-threaded multiple-pipelined (MTMP) processor architecture that tries to issue multiple instructions from multiple instruction threads in every clock cycle. For the performance evaluation, the paper proposes a modified analytic model that provides a quick prediction of utilization of pipelines. Unlike previous analytic models of multiple-threaded architecture, the model presented here concerns the utilization of multiple pipelines. It deals not only with pipeline dependencies but also with structure conflicts. The model can be used for turning processor parameters when a MTMP is designed.

Key Words

Multithreading, multipipelining, scheduling, speed-up, pipeline utilization

1. Introduction

The performance of single-threaded processors has been improved significantly by introducing deep pipelines [1] and by dispatching more than one instruction per clock cycle [2]. Because of the significant advances in circuit technology, more and more gates are available for designing multiple pipelined functional units in a single chip. However, the single-threaded processor will no longer significantly increase the processing speed because of the data and control dependencies between the instructions within a single thread. The dependencies introduce pipeline bubbles by forcing interlock delay between dependent instructions, and result in a low functional unit utilization.

The multiple-threaded multiple-pipelined (MTMP) architecture can make multiple pipelines busy and hence can further improve the processor performance. Multiple thread slots dispatch instructions from multiple threads simultaneously. A thread slot is a hardware unit that is responsible for fetching and decoding instruction. Usually, the number of thread slots is less than the number of instruction threads. The instructions are executed in functional units.

When the number of thread slots is given, using more functional units in a single processor will improve the processor performance but will result in a low functional unit utilization. On the other hand, using fewer functional units will improve utilization but at the cost of reduced performance. A compromise between processor performance and functional unit utilization must be made for the processor design. Dubey et.al. [3] proposed an analytic model for evaluating the multiple-threaded architecture. Their model is limited to predicting the performance of a processor with single pipeline and single thread slot. Because at most one instruction can be issued on every clock cycle, no structure conflict exists. The processor performance is affected only by the distribution of instruction interlock delay. When the processor has sufficient instruction threads for interleaved scheduling (for example, when the number of threads is equal to or larger than the maximal number of cycles required by execution pipeline stage), the processor utilization is said to be 100%. But when the processor has six independent execution pipelines, for instance, the average pipeline utilization is only 16.7%.

This paper proposes an analytic model that is used to quantify the utilization of multiple pipelines for MTMP architecture. The model deals not only with pipeline dependencies but also with structure conflicts. The effects of four important parameters S, T, E, and P $(STEP)$ will be evaluated where S is the the number of thread slots, T is the number of resident instruction threads, E is the maximal number of cycles required by execution stage, and P is the number of pipelined functional units. The model accepts a general distribution for the interlock delays with multiple latencies the same as in [3] and a general distribution for the different type of instructions that will be dis-

patched to different pipelines. The model predicts the utilization of multiple pipelines for different processor configurations, for example, for different number of thread slots, different number of pipelined functional units, and different number of resident threads.

The paper is organized as follows. Section 2 presents an architecture classification. Section 3 introduces the MTMP processor architecture. Section 4 describes the analytic model. Section 5 considers three examples in discussing the effects of STEP. Section 6 concludes the paper.

2. An Architecture Classification

According to the number of execution pipelines and the number of instruction threads, we divide the pipelined architectures into four types: *single-threaded single-pipelined* (STSP) architecture, *multiple-threaded single-pipelined* (MTSP) architecture, *single-threaded multiple-pipelined* (STMP) architecture, and *multiple-threaded multiple-pipelined* (MTMP) architecture (fig. 1).

Pipelining has been widely used in designing processors for exploiting the parallelism of operations. The potential speed-up of pipelining is equal to the number of pipeline stages used. This advantage encourages engineers to use deeper and deeper pipelines in designing high-performance processors. In the STSP architecture, a single thread is executed on a single pipeline (fig. 1(a)). The ideal throughput is at one instruction per cycle.

However, the ideal pipeline speed-up is rarely achieved in practice owing to the delays associated with pipeline dependencies and memory access latencies. NOOP instructions (no operation, that is, pipeline bubbles) will be inserted into the delay cycles. An approach for improving pipeline utilization is to multithread the processor. Such processors dispatch instructions from different threads on every clock cycle to tolerate the delays. We call them multiple-threaded single-pipelined architecture (fig. 1(b)). An instruction thread is defined as a set of instructions belonging to a particular context that can be executed independently of other instruction threads [4]. Because there is no pipeline dependency between instructions belonging to different threads, pipeline bubbles due to pipeline dependencies or processor stalls due to memory latencies can be prevented [5].

The STMP architecture is shown in fig. 1(c). Actually, In a practical processor there are dedicated pipelined functional units. Each functional unit performs a special operation so that the computing speed can be improved. In the modern high-performance microprocessor, the following pipelined functional units are in general use: ALU, shifter, branch unit, load/store unit, floating-point adder, multiplier, divider, and converter. Most commercial superscalar processors use the STMP architecture.

The MTSP architecture is efficient when the processor has a deep pipeline and the context switch overhead is low. In order to speed-up the context switch, multiple register sets and special data paths are usually needed to serve multiple threads. As only one thread uses its register set at a given time, the MTSP processors result in a low utilization of multiple register sets. Also notice that the average cycles per instruction cannot be less than one. On the other hand, the STMP architecture can improve the processing speed by providing effective execution pipelines. However, such single-threaded processor cannot use the multiple pipelines effectively owing to lack of sufficient instructions that can be issued on the same clock cycle. Usually, only a few pipelines are busy and the others are idle, even if the processor has superscalar capability [2]. Allocating multiple thread slots in a single processor, to realize multiple instruction threads to be executed simultaneously is a solution for improving the utilization of multiple pipelines [4, 6]. We call such architecture multiple-threaded multiple-pipelined architecture (fig. 1(d)). The multiple threads may be generated from a single program or from multiple programs. Thus, the MIMD parallel processing will be realized on a single processor.

3. The MTMP Processor Architecture Model

In the MTMP architecture, multiple thread slots, multiple dedicated pipelined functional units, and multiple register files are provided for executing multiple instruction threads in parallel. All the thread slots dispatch instructions on every clock cycle. Instructions are scheduled and issued to multiple functional units for execution, and results are written to register files.

Differing from a conventional pipelined processor, MTMP architecture contains state information of *T* threads. Each thread has its own program counter, status register, and register file. *P* functional units serve as *P* independent execution pipelines to support multiple instruction executions. There are *S* thread slots used for instruction dispatching. The instruction thread slots select *S* threads from *T* threads in an interleaved fashion. On every clock cycle, up to *S* instructions can be dispatched.

A separate instruction cache is provided for each of thread slots. In order to facilitate the interleaved thread selection, *T* instruction threads are distributed to *S* instruction caches equally. Each instruction cache contains an average of *T/S* instruction threads.

An *instruction scheduling unit* (ISU) schedules the *S* instructions and issues them to the functional units if there are neither structure conflicts among *S* instructions nor pipeline dependencies with previously issued instructions within a thread. If two or more instructions require the same functional unit on the same clock cycle, then structure conflict occurs.

The ISU selects one instruction to issue to the functional unit if dispatched instructions cause structure conflicts. In order to simplify the processor design, a simple instruction scheduling strategy, $round\ robin$, is employed. Availability of source operand is checked by using the *scoreboard* mechanism. If the scoreboard bits of the source operands are cleared, a ready instruction is found. Then the source operands are read out from the correct register file and destination register's scoreboard bit is set. The scoreboard bit will be cleared at the final clock cycle of execution stage. Thus the scoreboard bits could prevent

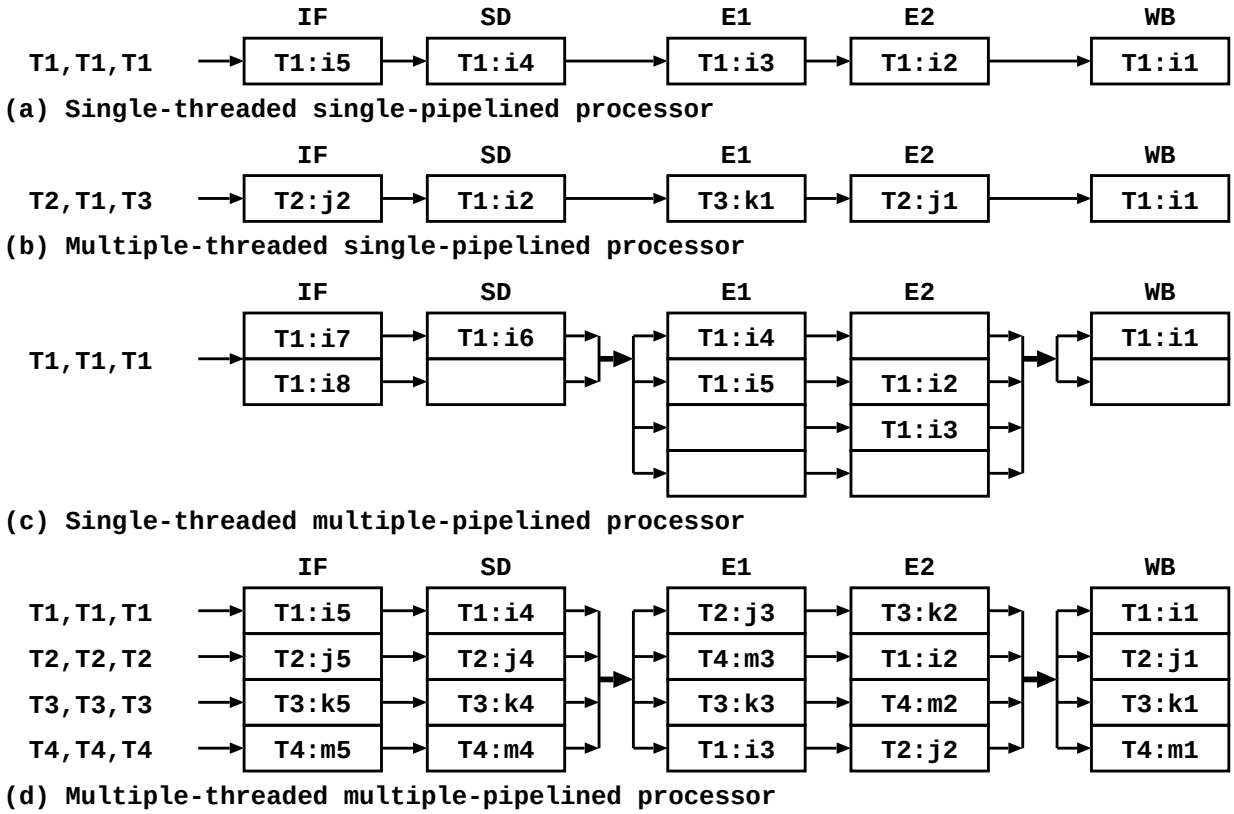

Figure 1. Pipelined execution models

incorrect data from entering into the pipelines.

The ISU is provided for each functional unit and FIFO registers for each thread slot are provided in the ISU. Un-issued instructions will be held in FIFO, waiting for scheduling in the following clock cycle. The thread slot is informed to stop fetching instructions from corresponding thread slots in the following clock cycle. Because the next instruction to the un-issued instruction is being fetched, the FIFO must have at least two registers for the thread slot. The total number of FIFO registers is $2 * S * P$, where S is the number of thread slots and P is the number of functional units.

The functional units carry out the desired data operations, and the results are written back into register file. Fig. 2 shows the data path and control path of the proposed MTMP architecture. An *interconnection network* (IN) is needed between the register files and the functional units. From the programmer's point of view, this physical MTMP processor is equal to S logical MTSP processors and each MTSP processor executes *T/S* threads concurrently.

Referring to fig. 2, each instruction pipeline comprises four stages: IF (instruction fetch), SD (schedule and decode), EX (execution), and WB (write back). During the IF-stage, each slot may fetch one instruction from dedicated cache. Because each slot deals with multiple instruction threads, the election of threads must be done before the fetching. The election strategy is very simple: rotating among all the threads (interleaved dispatching). The instruction thread slot may receive a "`freeze_fetching`" signal from the ISU. In this case, the instruction fetching will be frozen. Structure conflicts and source operand availability will be checked in the SD-stage. As mentioned above, the processor uses the round robin strategy to schedule the instructions. If an instruction is not issued to execution pipeline, it will be held in the two-word depth FIFO, and, the "`freeze_fetching`" signal will be sent to the corresponding instruction thread slot. Operands are also read in SD-stage. The interconnection network provides nonblocking paths for transferring data from the register files to the pipeline register. Because at most one instruction belonging to one thread could be executed, the register file was designed with only two read ports and one write port. The desired data operations are

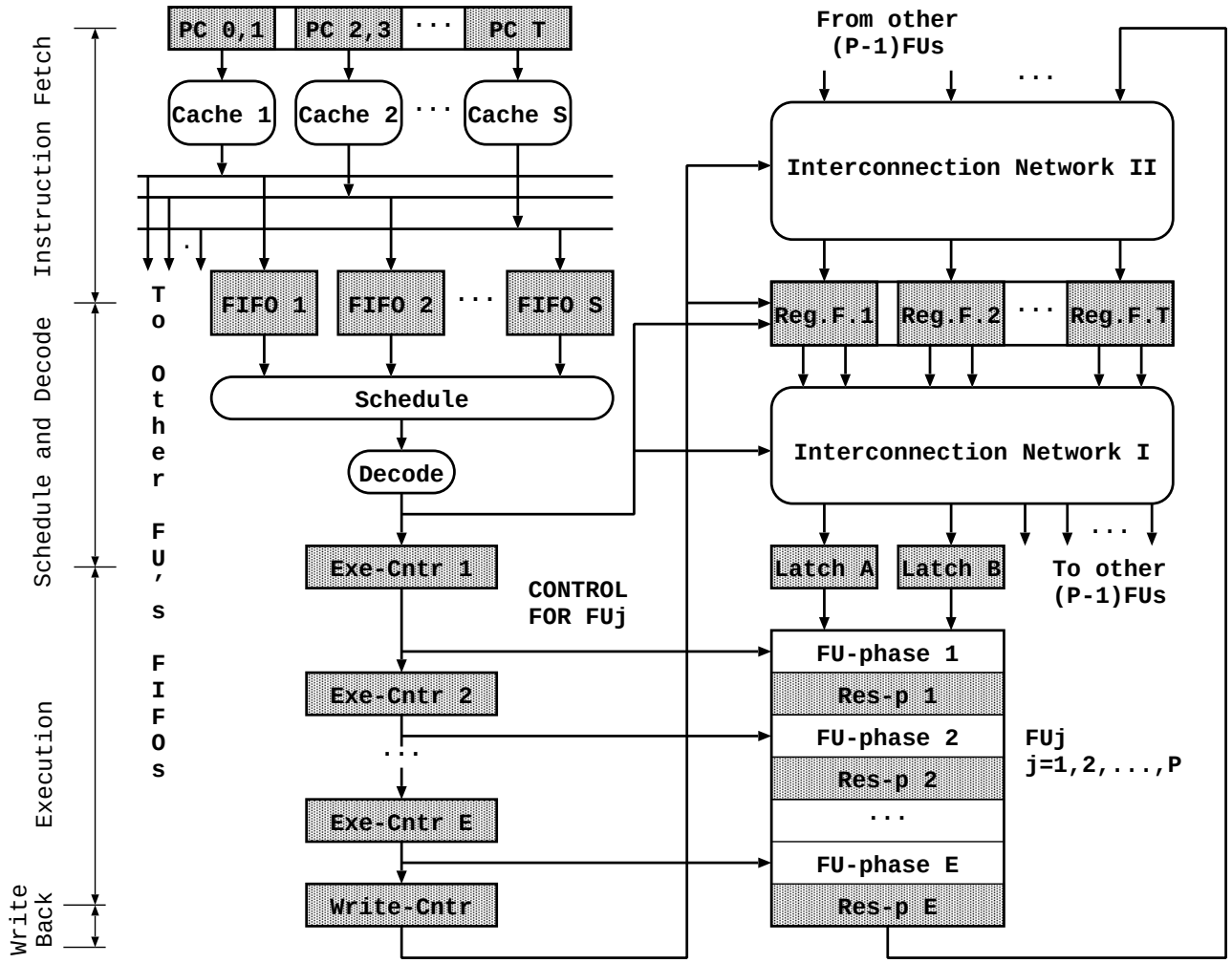

Figure 2. Data and control path in a MTMP processor

performed in the EX-stage. For most integer instructions, the execution stage takes one clock cycle. The floating point adder and multiplier take three clock cycles. And the floating point divider takes thirteen clock cycles. The results are written back into register file in the WB-stage. The interconnection network also provides nonblocking paths for transferring data from the result-registers to the register files.

4. The Analytic Model

In this section, we propose an analytic model for predicting processor-performance improvement and functional unit utilization for the MTMP processor architecture.

According to the MTMP architecture described in Section 3, we make the following assumptions: (1) There are *T* instruction threads. (2) There are *S* thread slots that can dispatch instructions simultaneously, and for fast instruction fetching, a separated instruction cache is provided for each thread slot. (3) The processor has more instruction threads than thread slots and the instruction threads are distributed equally to instruction caches: each cache holds average *T/S* threads. (4) The *T/S* instruction threads are interleaved and one instruction is fetched from the selected thread per clock cycle. (5) *P* pipelined functional units are provided. (6) All the functional units are effectively pipelined and are capable of accepting a new instruction in every cycle. (7) All the instructions are divided into *P* classes: the jth class instructions will be executed on the jth functional unit *(j=1,2,...,P)*. (8) The percentage of occurrences of the jth class instructions in dynamic instruction stream is ρ_j *(j=1,2,...,P)*. (9) the distribution of interlock delays is described by the probability vector $p = (p_1, p_2, ..., p_E)$, where p_j is the fraction of instructions that have an interlock delay of *j-1* cycles after they were scheduled, and *E-1* is maximum of interlock delay cycles, that is, *E* is the maximal number of cycles required by execution pipeline stage.

First of all, consider a conventional processor: *T*=1 and *S*=1. The *CPI* (cycles per instruction) estimate can be easily obtained [3] as:

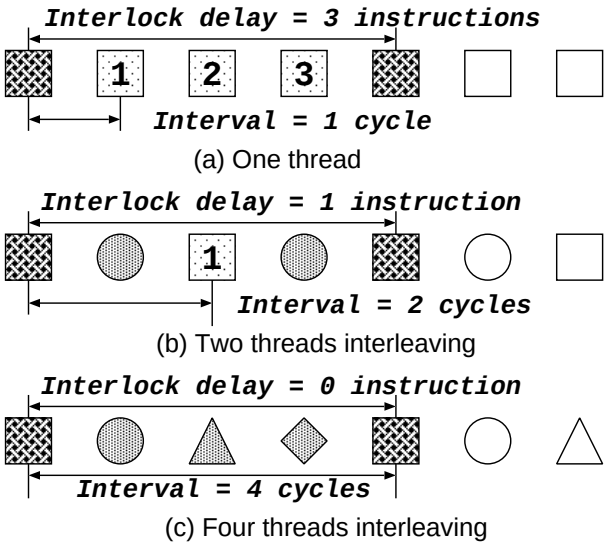

Figure 3. Interlock delay and interval cycles

$$CPI = 1 + \sum_{j=1}^{E} p_j * (j-1). \quad (1)$$

For example, an *E*=4 processor has following distribution of interlock delays. 20% of instructions in dynamic execution stream have an interlock delay of three cycles; 10% of instructions have an interlock delay of two cycles; and 30% of instructions have an interlock delay of one cycle. The remaining instructions require no interlock delay. The *CPI* of the processor is $1 + 0.4 * 0 + 0.3 * 1 + 0.1 * 2 + 0.2 * 3 = 2.1$, as given by (1). The number of real executed instruction-per-cycle is 1/2.1=0.476, that is, the processor utilization is 47.6%.

Now, consider that there are two instruction threads, *T*=2, that share the processor pipeline. Two threads are scheduled alternately, that is, the *interval* of dispatching instructions from a thread is two cycles (fig. 3). Because independent instructions will be inserted into the instructions of a thread, the suffering of interlock delay will be alleviated for each thread. For example, in the *E*=4 processor mentioned above, 20% of instructions have a new interlock delay of one instruction cycle.

Generally, in a processor with *T* instruction threads, the new value of interlock delays will be changed from $(j-1)$ cycles to $(j/T-1)$ cycles. Therefore, we can obtain the new *CPI*, denoted by C_T, for a *T-thread* processor as:

$$C_T = 1 + \sum_{j=1}^{E} p_j * max(0, (\frac{j}{T} - 1)). \quad (2)$$

In the mentioned *E*=4, *T*=2 processor, the C_2 is $1 + 0.4 * 0 + 0.3 * 0 + 0.1 * 0.5 + 0.2 * 1 = 1.25$. The number of real executed instructions per cycle is 1/1.25=0.8. The processor utilization increased from 47.6% to 80.0%, that is, 68% of improvement was achieved. Note that in the $T \geq E$ processor, C_T reaches the minimum of one cycle, that is, the processor utilization is 100%. However, if the processor has seven independent execution pipelines ($P = 7$), (ALU, shifter, load/store unit, branch unit, floating-point adder, multiplier, and divider, for instance), the total average utilization of all the pipelines must be $(100/7)\% = 14.3\%$.

For the $S > 1$ processor, maximal S instructions can be dispatched and issued to P execution pipelines. The structure conflicts must be considered in the $S > 1$ processor. According to the assumptions described in the beginning of this section, if the number of the jth class instructions to be dispatched on the same cycle is less than or equal to one, there will be no structure conflicts. If not, at most one instruction can be executed. The maximal average number of the jth class instructions $\xi_j(S)$, which are dispatched to jth functional unit, can be calculated by:

$$\xi_j(S) = \sum_{i=1}^{S} \frac{S!}{i!(S-i)!} \rho_j^i (1-\rho_j)^{S-i} * 1 = 1 - (1-\rho_j)^S. \quad (3)$$

The total number of instructions N, which are dispatched from S thread slots whose instructions are all data-ready, can be calculated by considering all the instruction classes:

$$N = \sum_{j=1}^{P} \xi_j(S) = \sum_{j=1}^{P} (1 - (1-\rho_j)^S). \quad (4)$$

N is an approximation derived from only resource constraints. We assume that the instructions come from S thread slots equally. Thus, each thread slot dispatches N/S instructions. Note that $N/S \leq 1$. Because each thread slot has T/S instruction threads and dispatches N/S instructions per cycle, the interval of dispatching instructions from a thread is equal to $(T/S)/(N/S)$, that is, there are $n = T/N$ virtual threads in a thread slot. In this case, similar to (2), the CPI for those N instructions, denoted by C_n, should be calculated by:.

$$C_n = 1 + \sum_{j=1}^{E} p_j * max(0, (\frac{j * N}{T} - 1)). \quad (5)$$

The total number of real executed instructions I can be calculated by dividing N by C_n, $I = N/C_n$, and the total average utilization of all the pipelines μ can be obtained by dividing I by P:

$$\mu = \frac{N}{P * C_n}. \quad (6)$$

The performance improvement is measured by the speed-up ratio ν, which is defined as the ratio of execution time required by $S > 1$-slot parallel multithreaded execution to those by $S =$ 1-slot concurrent multithreaded execution. The execution time of a given program can be expressed as the product of three terms: $i * c * t$, where i is the number of instructions required, c is the average number of cycles per instruction, and t is the

time per cycle. We can get the speed-up ratio ν from (7) at the assumption of same t in the both cases:

$$\nu = \frac{i * C_T * t}{i * (C_n/N) * t} = N * \frac{C_T}{C_n}. \quad (7)$$

5. Examples and Validation

As mentioned in Section 3, in the MTMP processor, there are four parameters that influence the processor performance and utilization of functional units. The four parameters are denoted by S, T, E, and P. S is the number of thread slots that affects the capability of dispatching instructions per cycle. T is the number of resident instruction threads, which affects the interval cycles of thread interleaving. E is the maximal number of cycles required by execution stage and it affects the interlock delay cycles, and P is the number of pipelined functional units, which affects the structure conflicts.

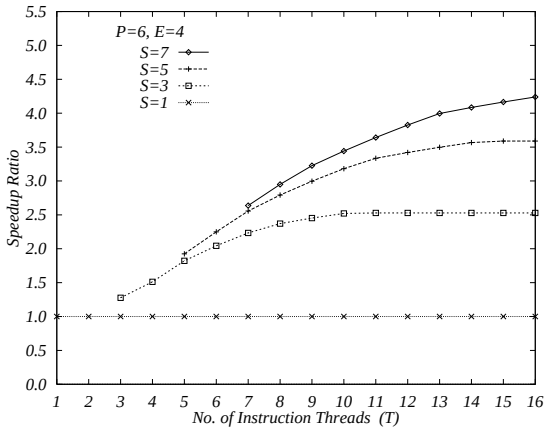

Figure 4. Speed-Up ratios for P=6 and E=4 processor

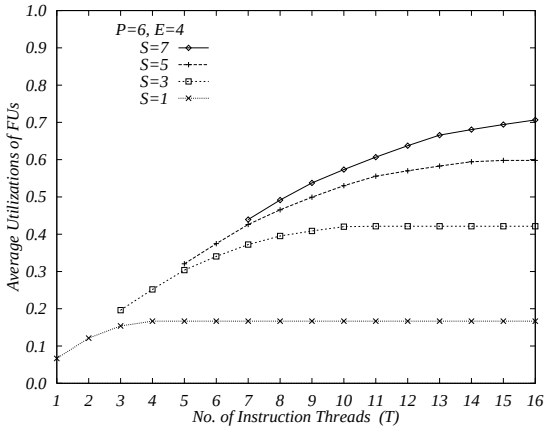

Figure 5. FU utilization for P=6 and E=4 processor

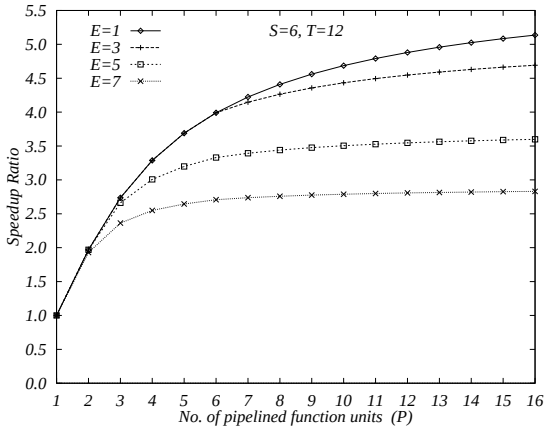

Figure 6. Speed-Up ratios for S=6 and T=12 processor

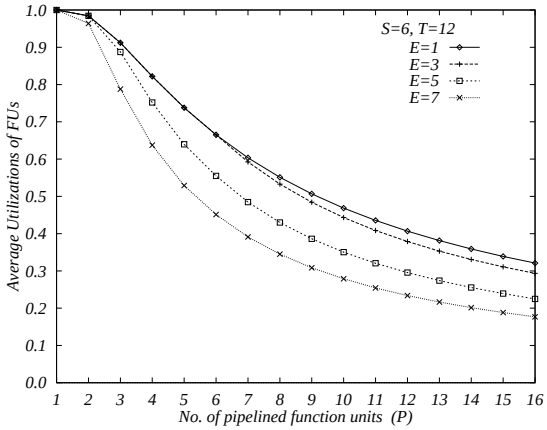

Figure 7. FU utilization for S=6 and T=12 processor

In the following examples, we assume that $\rho_j = 100\%/P$ for $j = 1, 2, ..., P$ and $p_j = 100\%/E$ for $j = 1, 2, ..., E$. The first example (fig. 4 and 5) shows the effects of S and T when P=6 and E=4. By increasing the number of thread slots S, we improve the speed-up and the average utilization, but when the number of thread slots is greater than four, significant improvement cannot be obtained by further increasing the number of thread slots. Also, we found that increasing the number of instruction threads T does not always result in performance improvement and utilization improvement.

The second example (fig. 6 and 7) shows the effects of P and E, when $S = 6$ and $T = 12$, on speed-up ratio and average utilization. When E is large, increasing the number of functional units results not in increased speed-up but in decreased utilization of functional units.

The third example (fig. 8 and 9) shows the effects of T and E, when $S = 6$ and $P = 6$. Note that there are upper bounds of speed-up ratio and average utilization that are caused by structure conflict. The upper bounds will be reached quickly when

E is small. In this case, increasing the number of instruction threads does not result in increased speed-up and increased utilization.

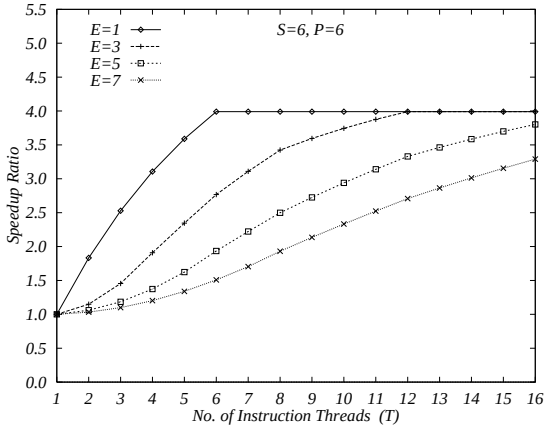

Figure 8. Speed-Up ratios for *S*=6 and *P*=6 processor

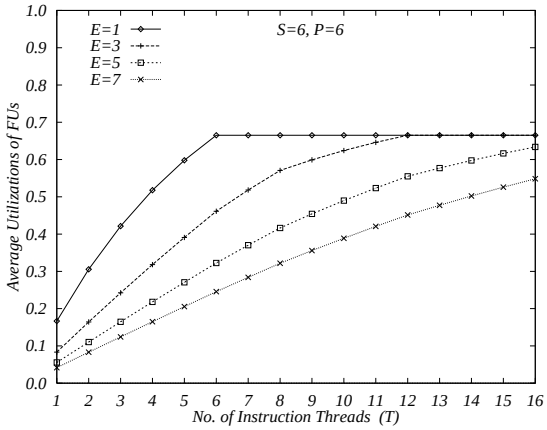

Figure 9. FU utilization for *S*=6 and *P*=6 processor

For the model validation, we choose a texture mapping program, that maps a texture pattern onto a 3D object's perspective projection in screen space. Seven functional units are arranged for executing the program: (1) integer unit performs integer arithmetic and logic operations; (2) load/store unit performs memory access; (3) branch unit evaluates condition codes and transfers control to new address; (4) floating-point adder performs floating-point add, subtract, and comparison; (5) floating-point multiplier performs floating-point multiplication; (6) floating-point divider performs floating-point division; and (7) floating-point convert unit performs floating-point/integer data type conversions.

We assume that all the functional units are capable of receiving a new instruction per cycle but have different execution cycles as follows. The integer unit and the branch unit have one execution cycle. The load/store unit has two execution cycles when the data cache hits. The floating-point adder, multiplier, and convert unit have three execution cycles. The floating-point divider has thirteen execution cycles.

The program is compiled to assembly code and the multiple code streams are used as inputs to a MTMP architecture simulator. As mentioned in Section 3, the processor has a separated instruction cache for each thread slot and a data cache for all the thread slots. The multiple streams are distributed equally to the instruction caches. In order to simplify the simulation, we assume that the cache accesses always hit.

For the calculation by the equations, we get the instruction distribution ρ_j for $j = 1, 2, ..., P$ and the distribution of interlock delay cycles p_i for $i = 1, 2, ..., max(E_1, E_2, ..., E_P)$ from the dynamic execution stream. The simulated results of speed-up ratio on texture mapping program are almost equal to the results generated by using analytic model, with the average deviation less than 1%.

6. Conclusion

In this paper, we have presented a multiple-threaded multiple-pipelined architecture that realizes multiple multiple-threaded single-pipelined processors in a single processor environment. The MTMP processor contains multiple pipelined functional units and multiple thread slots. Each thread slot has multiple instruction threads, and the threads are interleaved for dispatching. In order to evaluate the MTMP architecture, we proposed an analytic model that provides a quick prediction for performance improvement and for average utilization of multiple functional units. The model deals not only with pipeline dependencies but also with structure conflicts. The effects of four important parameters in MTMP architecture, $STEP$, were evaluated and the analytic model was validated by simulation.

References

[1] "R4000: 64-bit RISC microprocessor user manual," (Toshiba Corp., Tokyo, Japan, 1992).

[2] K. Diefendorff and M. Allen,"Organization of the Motorola 88110 Superscalar RISC Microprocessor," in *IEEE MICRO*, 12(2), April 1992, pp40-63.

[3] P. K. Dubey, A. Krishna, and M. J. Flynn, "Analytical modeling of multithreaded pipelined performance," in *Proc. of the 27th Annual Hawaii Intl. Conf. on System Sciences*, Maui, Hawaii, 1994, pp361-367.

[4] R. Guru Prasadh and Chuan-lin Wu, "A benchmark evaluation of a multi-threaded RISC processor architecture," in *Proc. of the 20th Intl. Conf. on Parallel Processing*, Boca Raton, FL, 1991, ppI:84-91.

[5] D. C. McCrackin, "Eliminating interlocks in deeply pipelined processors by delay enforced multistreaming," *IEEE Trans. on Computers*, 40(10), Oct. 1991, pp1125-1132.

[6] H. Hirata, K. Kimura, S. Nagamine, Y. Mochizaki, A. Nishimura, Y. Nakase, and T. Nishizawa, "An elementary processor architecture with simultaneous instruction issuing from multiple threads," in *Proc. of the 19th Annual Intl. Conf. on Computer Architecture*, Gold Coast, Australia, May 1992, pp136-145.