Theoretical Evaluation of Simultaneous Multithreading Parallel Queue Processor Architecture

Hirotoshi Sasaki[1], Yoshitomo Okumura[2], Ben A. Abderazek[1],
Soichi Shigeta[1], Tsutomu Yoshinaga[1] and Masahiro Sowa[1]
[1]Graduate School of Information Systems, the University of Electro-Communications
1-5-1, Chofugaoka, Chofu-City, Tokyo, 182-8585, JAPAN
Tel.: +81-424-43-5697, Fax: +81-424-43-5681
[2]Presently, SONY Corporation, JAPAN
E-mail: sasaki@sowa.is.ac.jp, {ben, shigeta, yosinaga, sowa}@is.uec.ac.jp

Abstract: In this paper, we describe abstraction of Simultaneous Multithreading Parallel Queue Processor (SMT-PQP) architecture. SMT-PQP is based on the queue calculation model. SMT-PQP can provide both Instruction Level Parallelism (ILP) and Thread Level Parallelism (TLP). Namely, it has the ability to offer an appropriate granularity of parallelism for all applications. As a result, SMT-PQP achieves high usability of calculation resources, it expands register resources and does not need to support a register renaming function. A theoretical evaluation of SMT-PQP is described.

1. Introduction

Nowadays, the most widely known processor technology is used for computing products (notebooks, tablet PCs, servers, etc.), mobile/handheld devices, robot control devices, and so on. As the demand for increased performance in microprocessor continues, a new processor is requested to meet this demand. The performance of existing processors increases due to technology improvements, which are unfortunately reaching their limit. This is one of the reasons why existing processors use Random Access Memory (RAM) for intermediate results. An instruction needs to appoint a register to use for reservation statement in RAM model. The problem is that the trade-off depends on the number of registers and the instruction length. As a result, this problem causes hardware complexity and low performance.

Accordingly, we do research on a processor architecture which is different from existing processors' architectures. This architecture uses the queue calculation model [1]. We proposed several queue processors which store intermediate results in a queue [2]. As a next step, we study high performance processor architecture. The new processor is Simultaneous Multithreading Parallel Queue Processor (SMT-PQP). SMT-PQP is Parallel Queue Processor (PQP) using queue calculation model. It uses queue registers as registers for intermediate results. The conventional PQP can achieve only ILP, and it cannot entirely utilize the possible parallelism of an application program. Therefore, we add SMT technology [3]. Thus, SMT-PQP is able to provide TLP. Furthermore, this processor can use both parallelisms without the extra hardware for register renaming. ILP is supported by exploiting a parallelism of superscalar processor and TLP is supported by exploiting a parallelism of multi-thread processor. Naturally, SMT-PQP has all the advantages of PQP by using queue calculation model. In this paper, we describe a theoretical performance evaluation of this processor.

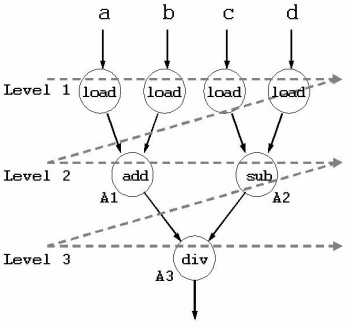
2. SMT-PQP Fundamentals

2.1 Queue calculation model

The basic principle of SMT-PQP is the queue calculation model. In this model, intermediate calculation results are stored in a memory area with First In First Out (FIFO) structure. This memory is called queue. A queue consists of concatenated words. All operands load data from the head of the queue (QH: queue head pointer) and store results at the tail of the queue (QT: queue tail pointer). Therefore, the queue calculation model uses the queue as a memory for intermediate results, it does not need to specify the queue in machine instructions and all the parallelism, that a program has, is easily utilized.

Figure 1 shows a syntax tree of the expression (a+b)/(c-d) in the queue calculation model. The arithmetic operations correspond to the internal nodes of the tree and the fetch operations correspond to the leaf nodes. The level order scan traversal (LOST) is done by visiting the nodes of the binary tree from the deepest to the shallowest levels and from the left to the right within each level. The nodes of each level form a queue. Actually, these level-based queues are connected in a single queue.

Figure 2 shows the queue during execution. It represents the state of the queue during the actual calculation of the expression shown in Figure 1. The operations of queue registers are determined by operands placed in the



Expression: (a+b)/(c-d)

Figure 1 Syntax tree for queue calculation model

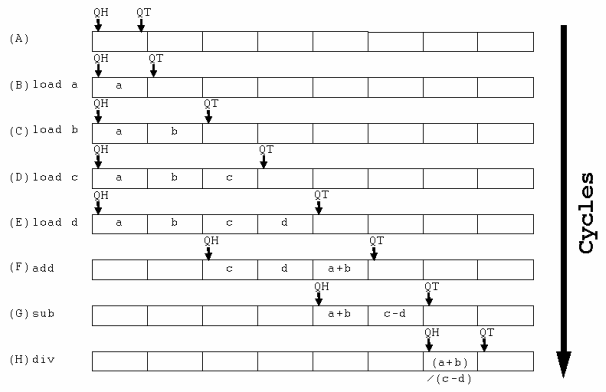


Figure 2 Contents of queue during calculation

queue. These operands are taken from the queue head, and the results of the calculations are written to the place pointed by QT. The initial state (Figure 2 (A)) is the empty queue. “a” is inserted to the tail of the queue (Figure 2 (B)), then “b”, “c”, “d” are inserted (Figure 2 (C), (D), (E)). Next, “a” and “b” are picked up by instruction A1, they are added, and the calculation result “a+b” is inserted at the tail (Figure 2 (F)). In a similar way, “c” and “d” are picked up by instruction A2, subtracted, and the result “c-d” is inserted at the tail (Figure 2 (G)). Then, “a+b” and “c-d” are picked up by instruction A3, they are processed, and the calculation result “(a+b)/(c-d)” is inserted at the tail (Figure 2 (H)).

This queue calculation model makes parallel execution easy. In Figure 1, individual queues are used by each level. Parallel processing of instructions is obtained by allowing the four “load“ instructions from level 1 to be processed at the same time, and “add“ and “sub“ from level 2 to be executed simultaneously. If this calculation is done in serial, it will take seven steps. Using simultaneous calculation (i.e. parallel execution) reduces the number of steps to three.

2.2 Use of ILP and TLP

SMT-PQP combines two properties - shared calculation resources and application of ILP and TLP without physically building internal resources. For example, when a program has low TLP and high ILP, SMT-PQP is able to support the ILP by using one long queue. On the other hand, if the program has high TLP and low ILP, SMT-PQP can support the TLP by using several short queues. Therefore, the designed processor is suitable for both ILP and TLP.

2.3 No need register renaming

SMT-PQP uses a shared queue as registers for intermediate results without register renaming. Moreover, the processor has a shared calculation resource because it is based on the SMT technology. SMT-PQP can expand register resources while keeping the registers’ names unique. There is no possibility for register name collision. Therefore, SMT-PQP is not required to perform register renaming like the conventional processor and the register renaming function becomes useless.

Shared register resources shows that an application program can be executed with both high ILP and high TLP.

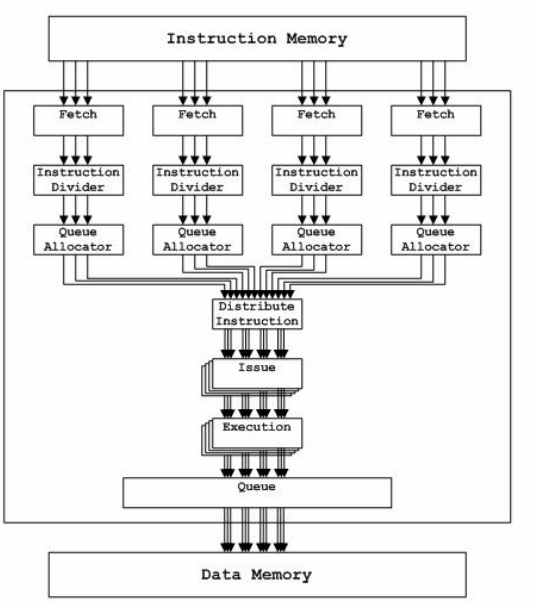


Figure 3 SMT-PQP architecture

Although the hardware of a SMT processor is complex, the hardware of SMT-PQP is simpler.

3. Design

3.1 Processor architecture

SMT-PQP consists of four Processor Elements (PEs). Each PE can work in parallel, and each thread can operate in parallel. Also, SMT-PQP can perform both single-thread execution and multi-thread execution.

In this paper, we discuss parallel thread executions by four PEs. Stages constituting a pipeline are assigned to each PE. A PE can process three instructions in parallel. Therefore, a parallel execution of maximum 12 instructions is possible. Physical registers for intermediate results are contained in a single queue. When SMT-PQP operates a program of high TLP, one physical queue is divided into several logical queues that are assigned to the threads. Thus each PE gives an independent short queue to each of its threads. When SMT-PQP operates a program of high ILP, the queue is not divided, but is used as one long queue.

Figure 3 shows SMT-PQP architecture. The pipeline of SMT-PQP consists of six stages. During the Fetch stage instructions are fetched from instruction-memory. Then they are divided in the Instruction Divider stage, decoded and placed in the queue during the Queue Allocator stage. Next, in the Distribute Instruction stage, instructions are distributed to each Issue stages.

Table 1 Execution resources

Unit	Operation	Number
ALU	add/sub/shift/logical operation	4
Multiplier	multiply	4
Divider	divide	4
Memory Access	load/store	4
1Data	immediate access/increment/decrement	4

The number of the Issue stages is same as that of the Execution stages. Issue stages issues instructions to the Execution stages, where they are processed and intermediate results are stored to the queue.

SMT-PQP has six stages. Three of them are independent:

- Fetch,
- Instruction Divider,
- Queue Allocator,

and, three are shared:

- Distribute Instruction,
- Issue,
- Execution.

The hardware specification of SMT-PQP is as follows:

- 4 processing elements (PEs)
- 3-instruction fetch width per PE
- 4-instruction issue width
- 128-word queue
- 1- or 3-byte instructions
- 32-bit memory address space

There are four Stall Controllers in SMT-PQP that control the pipeline. A Stall Controller observes Fetch, Instruction Divider and Queue Allocator stages. If SMT-PQP needs to become in stall state, the Stall Controller sends a stall signal to all units. Also, Stall Controllers clear buffers and restart thread processing. They take an important role of controlling units.

Table 1 shows the contents of the execution resources. Performance depends on the number of execution resources in a processor. If the twelve fetching instructions are of the same kind, it will take three steps to complete processing. The maximal number of processing instructions in one execution function is 4. The greatest performance is achieved when all execution resources are used. However this makes the hardware more complex.

Except for the units shown in Table 1, there are other controllers - Frame, Thread and Fission/Fusion controllers.

3.2 Design of an effective register resources usage

SMT-PQP issues a fission instruction in the thread controller to split queues. The divided queues are convenient for improving efficiency. A thread uses one of the queues assigned to PE.

When treads finish, PE executes a fusion instruction to combine the queues in a way that the queue registers return one queue again. Therefore, SMT-PQP can operate with both split and non-split queues.

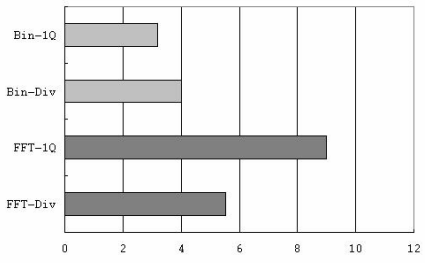


Figure 4 Instruction per cycle

In a single-thread operation the whole queue is assigned to the PE without splitting. This is done in order to effectively use all register resources. The queue is treated as one queue - there is no need to split it, because there is only one thread. Thus, SMT-PQP can use much more register resources.

4. Evaluation

4.1 Evaluation program

For evaluation we used two programs: `Bin`, which executes binary operations; and `FFT`, which implements a Fast Fourier Transformation. SMT-PQP runs these programs in a multi-thread mode.

`Bin` is a small program. It repeats a binary operation four times. It is scheduled in two ways – `Bin-1Q` uses one long queue, and `Bin-Div` uses several small queues. The latter scheduling, `Bin-Div`, is suitable for achieving high TLP.

`FFT` program is a typical program of signal processing. It is tuned for eight points butterfly computation. Similarity to the `Bin` program, `FFT` is scheduled in two ways: `FFT–1Q` and `FFT–Div`. Because `FFT` has a possibility of large ILP and TLP, we examine the ILP for `FFT-1Q` and the TLP for `FFT-Div`. We also pay attention that `FFT-Div` performance can be easily affected by thread issues overhead.

4.2 Result and Consideration

Figure 4 shows instruction per cycle for `Bin–1Q`, `Bin–Div`, `FFT–1Q` and `FFT–Div` executed in multi-thread mode. This is the average IPC measured for each of the four programs.

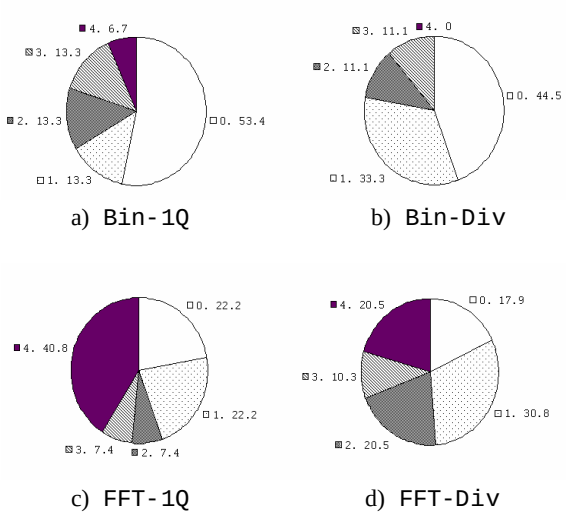
`Bin–Div` has a slightly better IPC than `Bin–1Q`. The reason for this is that the scheduler can perform better scheduling for `Bin-Div` by efficient use of all PEs. On the other hand, `Bin–1Q` can use a higher ILP, because it is a simple program and higher TLP is achieved harder than higher ILP.

The IPC of `FFT–1Q` is better than the one of `FFT–Div`. The theoretical maximum ILP is 12 instructions per cycle. The achieved ILP is 9.0, which is the best for all tests, because `FFT–1Q` can effectively use available PEs.

Because of the multi-thread execution `FFT-Div` uses mainly TLP and sometimes ILP. Each PE is responsible for the processing of two points. Thus all four PEs handle in parallel the 8 points of the FFT. However there is an overhead caused by the fission instruction, so the IPC of `FFT–Div` is 5.5.

If `FFT–Div` is scheduled for a better TLP, the IPC becomes lower. The reason for this result is in the structure of the issue controller. Threads are organized in a ring. The time for thread management is not short. Therefore, the overall execution is affected by the sequential thread switching.

When `FFT` is scheduled for a single-thread execution, the IPC is 3.0, which is about 1.8 times smaller than the IPC of an optimally scheduled `FFT-Div`. The final result is that



a) `Bin-1Q` b) `Bin-Div`

c) `FFT-1Q` d) `FFT-Div`

Figure 5 Ratio of execution resources activity [%]

`FFT-Div` can use both TLP and ILP.

Figure 5 shows the ratio of the execution resources activity. Tested programs use only three types of execution resources: ALU, Multiplier and 1Data. Each of them has 4 instances. At each cycle each type of resource may have 0, 1, 2, 3 or 4 active instances. The labels in Figure 5 stand for the number of resources with the corresponding number of active instances.

For Figure 5 a), the ratio of zero activity is the highest of all – 53.4%. This means that more than half of the time ALU, Multiplier or 1Data had no active instances. So, `Bin-1Q` cannot effectively use execution resources. The ratio of maximum activity is 6.7%, which means that it is not possible to achieve high ILP (this is due mainly to the small size of the program).

For Figure 5 b), the ratio of zero activity is better than that of `Bin-1Q`. `Bin-Div` executes optimized `Bin` on four threads. As a result, this scheduling is more effective, but it does not load all resources (the ratio of resources with full instance activity is 0%).

For Figure 5 c), the ratio of 4-instance activity is the highest of all. `FFT-1Q` can use ILP very well. This shows that the processing is very efficient. Since this program reaches the maximum ILP, which SMT-PQP can provide, the execution makes use of all the parallelism.

For Figure 5 d), the ratio of zero activity is the lowest of all. Also, the 4-instance activity ratio is lower than that of `FFT-1Q`. `FFT-Div` tries to employ TLP. However, because of the scheduling of `FFT`, it is not possible to effectively use high TLP. So, in SMT-PQP, `FFT` did not reach the maximal TLP utilization.

Figure 6 shows the average register usability. Both `Bin-1Q` and `FFT-1Q` have higher register usability than their `Div` counterparts. Since ILP occurs much more often than TLP, `1Q` scheduling can benefit from this, while `Div` scheduling cannot fully utilize TLP. If we would like to improve the performance of SMT-PQP, we must find a better method for thread management in order to perform effective use of register resources.

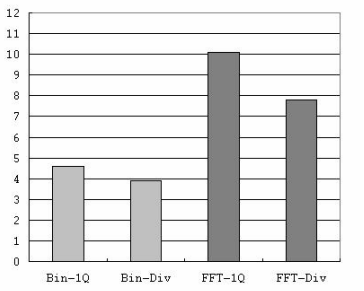


Figure 6 Average register usability [%]

These results become possible because of the nature of shared register resources in SMT-PQP processor. Since calculations are based on breadth-first search, the performance of SMT-PQP is determined by the average register usability. Thus, SMT-PQP executes programs by using multiple split queues as well as by using a single queue. In such a way the processor can achieve both high ILP and TLP.

5. Conclusions

In this paper, we describe a theoretical evaluation of SMT-PQP in order to confirm its performance. The main goals are two. First, SMT-PQP can achieve both ILP and TLP. Second, it can effectively work with shared queue register resources. The evaluation shows the ratio of the execution resources activity and the average register usability. We use these results to verify SMT-PQP performance.

In Chip Multiprocessing technology TLP is realized by having multiple threads executed on increased number of processors. This is not needed for SMT-PQP, because several threads can be executed simultaneously, while calculation resources are still effectively used.

Furthermore, SMT-PQP is suitable for both single- and multi-thread executions. In a single-thread execution, the queue register operates as one long queue, and effective use of calculation resources increases ILP. On a multi-thread execution, the queue register contains several small logical queues, and it is possible to achieve high TLP. Finally, SMT-PQP does not require the use of a register renaming technique like it is needed for conventional SMT processors. So, SMT-PQP does not need a register renaming function.

Reference

[1] B. Preiss, V. Hamacher: “Data Flow on a Queue Machine”, 12th Int. IEEE Symposium on Computer Architecture, pp.342-351, (1985).

[2] M. Sowa, B. Abderazek, S. Shigeta, K. Nikolova, T. Yoshinaga: “Proposal and Design of a Parallel Queue Processor Architecture”, 14th IASTED PDCS2002, pp.554-560, (2002).

[3] S. Eggers, J. Emer, H. Levy, J. Lo, R. Stamm, D. Tullsen: “Simultaneous Multithreading: A Platform for Next-Generation Processors”, IEEE Micro, pp.12-19, (1997).