The Instruction Execution Mechanism for Responsive Multithreaded Processor

Tstomu Itou
School of Science for
Open and Environmental Systems
Keio University
Yokohama, Kanagawa,
223-8522, Japan
itou@ny.ics.keio.ac.jp

Nobuyuki Yamasaki
Department of Information
and Computer Science
Keio University
Yokohama, Kanagawa
223-8522, Japan
yamasaki@ics.keio.ac.jp

Abstract

This paper describes the instruction execution mechanism of *Responsive Multithreaded (RMT) Processor* for distributed real-time processing. The execution order of each thread is controlled by using priority in *RMT Processor*. The highest priority thread is executed first in *RMT Processor*.

Real-time applications, such as soft real-time processing including multimedia processing, require high computing performance. So we design the vector processing unit. Since multiple threads are executed in parallel by the multithreading architecture, these threads execute vector operations in parallel. We design the vector processing unit so that multiple threads are able to share vector registers and execute vector operations efficiently. Moreover, we design a vector compound execution mechanism to improve the performance of vector operations.

1 Introduction

Many processes with various timing constraints are executed in a real-time system. In order to guarantee these timing constraints, these processes have priority given by Real-Time Operating System (RT-OS) to control the execution order of each process. RT-OS executes the processes in the order of priority. When RT-OS switches the executing process, the overhead occurs. The hardware support is necessary to reduce this overhead in real-time systems.

We designed *Responsive Multithreaded (RMT) Processor* so that it can execute processes in real-time by hardware. *RMT Processor* executes the higher priority thread in first. We also design the vector processing unit so that it can achieve the high computation performance required for soft real-time processing. In this paper, we describe the instruction execution mechanism of *RMT Processor*.

2 Background

Responsive Multithreaded (RMT) Processor is a system-on-a-chip that integrates a processing unit(*RMT Processing Unit (RMT PU)*), *Responsive Link*[1] and various I/Os. It can support the distributed real-time processing by hardware.

2.1 Simultaneous Multithreading

The architecture of *RMT PU* is based on Simultaneous Multithreading (SMT) mechanism[2, 3]. Multiple threads are executed in parallel in the SMT architecture. Multiple instructions from multiple threads are fetched and issued in a clock cycle. Instructions are selected from different threads so that the dependence of instructions can be decreased and Instruction Level Parallelism (ILP) can be improved. Moreover, even when a processor executes a long latency instruction, this latency can be hidden by executing another thread.

2.2 Conflict Arbitration of Computation Resources by Using Priority

As SMT processor executes multiple threads in parallel, conflicts of computation resources, such as fetch slots, issue slots, cache access, and arithmetic logical units, occur. *RMT PU* arbitrates these conflicts by using priority. Each thread specifies priority in a dedicated register of *RMT PU*. When a conflict occurs, *RMT PU* selects the instruction of the highest priority thread and executes it first[4], so that *RMT Processing Unit* can control the execution order of each thread without RT-OS help. Since *RMT PU* has SMT architecture, it can reduce the overhead of context switching.

2.3 Context Cache

RMT PU has up to eight threads as hardware contexts including complete register set, PC, and status registers. If there are nine or more executing threads, the help of software is needed to exchange a hardware context. When the software exchanges a hardware context, it saves the context to the memory and restore the new context from the memory. So the overhead of context switching occurs. This overhead is a big problem in real-time systems. We designed a dedicate on-chip cache in *RMT PU* to save these contexts. This cache is connected to register files with wide bus. As context switching is performed by using this cache, the overhead of context switching reduces greatly.

2.4 Computing Performance of RMT Processing Unit

Since multiple threads are executed in parallel in the SMT architecture, the total performance can increase. But the performance of each thread may decrease for conflicts of computation resources. It doesn't meet the demand of computing performance required for soft real-time processing. So we design the instruction execution mechanism of *RMT PU* so as to achieve this high computing performance.

3 Design and Implementation

3.1 Design Plan

The data level parallelism (DLP) can be used by soft real-time processing. Same operations are repeated with a lot of data in soft real-time processing. There are two methods to achieve the high computing performance by repeating the same operations.

- SIMD Operation
- Vector Operation

On one hand, the SIMD operation is performed with the short latency, but data parallelism is small. On the other hand, the vector operation is performed with the long latency, but data parallelism is large.

Here, the long latency of the instruction is hidden by the multithreading architecture in *RMT PU*. So the latency of a vector operation can also be hidden. Instruction fetch slots and issue slots are shared by multiple threads in *RMT PU*. We consider that the large data parallelism of the vector operation is effective to reduce conflicts of the fetch slots and issue slots. Therefore we decide the vector operation for the instruction execution mechanism of *RMT PU*.

The block diagram of *RMT Processor* is shown in Figure 1.

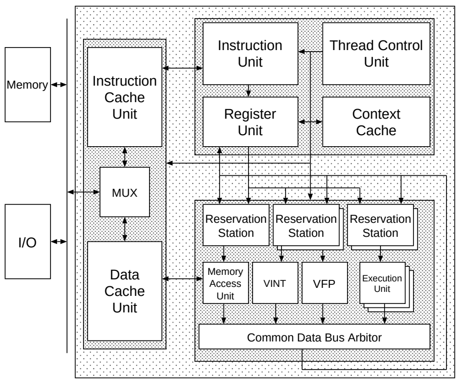

Figure 1: The block diagram of *RMT Processor*.

The thread control unit has a status register of each thread including executable, stopping, and priority. It controls the context cache to save or restore threads. The context cache consists of on-chip memory for saving the contexts. The instruction unit selects the issued instruction according to priority of each thread, and issues the instructions to reservation stations. Priority of each thread is also used in cache systems, reservation stations, operation units, and reorder buffers to arbitrate conflicts. We designed vector processing units such as VINT (vector integer unit) and VFP (vector floating-point unit).

3.2 Vector Processing Units

The block diagram of the vector processing unit is shown in Figure 2.

The vector control unit performs effective address calculation for accessing to vector registers, reserves and releases vector registers, and executes vector compound operations, which are described later. In order to execute vector operations of multiple threads in parallel, a vector processing unit has two operation pipelines (vector execution units). Each integer oper-

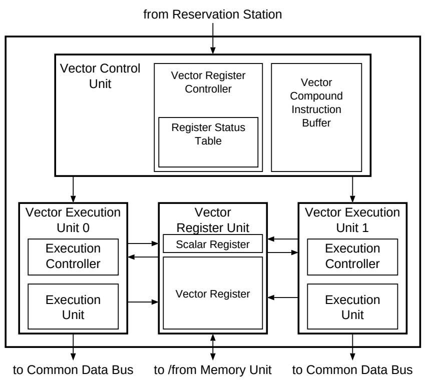

Figure 2: The block diagram of a vector processing unit.

ation pipeline has eight operation units so that the integer operation pipeline executes eight vector elements in parallel to increase throughput of vector operations. Each floating point operation pipeline has four operation units so that the floating point operation pipeline executes four vector elements in parallel.

3.3 Reserving and Releasing Vector Registers

Since *RMT PU* can execute multiple threads in parallel, these threads may execute vector operations at a time. If each thread has own vector registers, the hardware amount will become large. So we design vector registers that can be shared by multiple threads. When each thread executes vector operations, it reserves vector registers first and executes vector operations subsequently. If the thread finishes vector operations and doesn't use vector registers any more, it releases the vector registers. Thereby, another thread can reserve vector registers and execute vector operations at the same time. From the point of the trade-off with the amount of transistor, 512 words are implemented to the vector integer unit and the vector floating-point unit respectively.

The configuration of vector registers, such as element length and register size is different depending on applications. In order to share vector registers efficiently, it is necessary to allocate vector registers with the suitable size. So each thread specifies the required size of vector registers when it reserves vector registers. If unused vector registers are enough, the vector control unit allocates the specified size of vector registers to the thread. If vector registers are not enough, the reserving operation is failed. The configuration of the vector register which contains the allocated area, vector length, register size is saved in a register status table.

When the vector control unit receives a vector operation, it calculates effective address of the vector register with the configuration information in this table. The vector execution pipeline uses this address to access the vector registers.

If variable size can be specified at reserving vector registers, hardware logic will become complex and the amount of transistor will increase. Additionally, fragmentation will occur if reserving and releasing are repeated. So we limit the configuration specified in the reserving operation. The configuration that can be selected is shown in Figure 3.

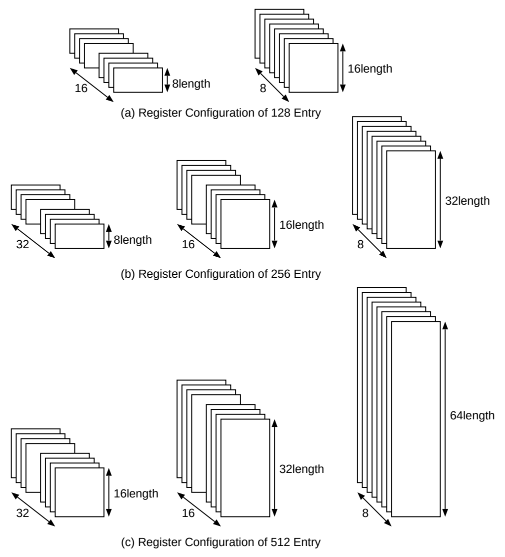
16
8length
8
16length
(a) Register Configuration of 128 Entry
32
8length
16
16length
8
32length
(b) Register Configuration of 256 Entry
32
16length
16
32length
8
64length
(c) Register Configuration of 512 Entry

Figure 3: The configuration of vector resisters.

We prepare two new instructions to execute reserving and releasing operation. VRES (Vector REServe) instruction executes reserving vector registers. It specifies the configuration of vector registers. VREL (Vector RELease) instruction executes releasing vector registers.

3.4 Vector Compound Operation

The same operations, such as multiply-and-add operation, are repeated in many soft real-time processing. We design the vector compound operation mechanism which executes a series of vector operations performed repeatedly. A programmer defines a series of vector operations as a vector compound operation.

These series of vector operations are executed as one instruction. When the vector control unit receives a vector compound operation, it executes a series of vector operations defined by a programmer continuously.

4 Evaluation

We evaluate the vector processing mechanism by using the program that executes an IDCT of 8×8 array. The configuration of vector registers is 128 registers with 8 vector length.

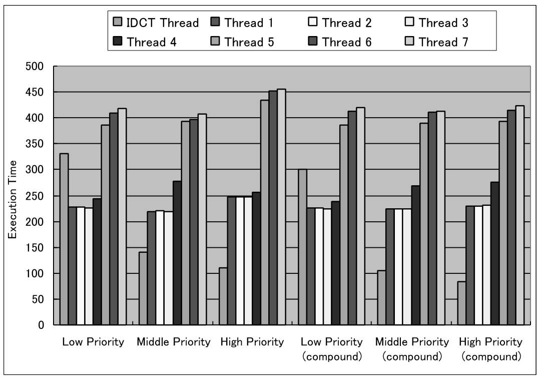

Figure 4: The execution time with using priority.

RMT PU can control execution order of each thread by using priority. Figure 4 shows execution time in which each thread was controlled by priority. In this figure, the IDCT thread executed IDCT program with vector operations. The other threads executed integer sort program as CPU load. The Thread1 was the highest priority thread and the Thread7 was the lowest priority thread. Low priority shows that the IDCT thread had the lowest priority among all threads. High priority shows that the IDCT thread had highest priority. The "compound" shows that the IDCT program used vector compound operations. We defined eight multiply-and-accumulate operations as one compound operation.

The execution time of the IDCT thread which used compound operations decreases compared with not using compound operations. As a compound operation can perform many operations with one instruction, the utilization of the vector processing unit increased. When the IDCT thread has highest priority, the execution time of the other threads decrease too. As the number of vector instructions decreased by compound operations, conflicts of the fetch and issue slots decreased.

5 Conclusion

In this paper, we describe the design and implementation of the instruction execution mechanism for *Responsive Multithreaded Processor*. We design the vector operation mechanism to achieve high computing performance required by soft real-time processing.

Vector registers are shared by multiple threads, so that it executes vector operations of multiple threads efficiently without increasing amount of transistors. Additionally a vector compound operation mechanism performs multiple vector operations with one instruction, so that the utilization of the vector processing unit increases and the computing performance increases.

Acknowledgements

This study was performed through Special Coordination Funds of the Ministry of Education, Culture, Sports, Science and Technology of the Japanese Government.

References

[1] Nobuyuki Yamasaki. Design and implementation of responsive processor for parallel/distributed control and its development environment. *Journal of Robotics and Mechatronics*, 13(2):125–133, 2001.

[2] Susan J. Eggers, Joel S. Emer, Levy Henry M, Jack K. Lo, Rebecca L. Stamm, and Thllsen Dean M. Simultaneous multithreading : A platform for next-generation processors. *IEEE Micro*, 17(5):12–19, 1997.

[3] Dean M. Tullsen, Susan J. Eggers, Joel S. Emer, Levy Henry M, Jack K. Lo, and Rebecca L. Stamm. Exploiting choice: Instruction fetch and issue on an implementable simultaneous multithreading processor. In *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, 1996.

[4] Masato Utiyama, Tsutomu Itou, Junichi Sato, Nobuyuki Yamasaki, and Yuichiro Anzai. A new processor architecture for real-time systems. In *IFAC Conference on New Technologies for Computer Control 2001*, 2001.