

Towards Reconfigurable Cache Memory for a Multithreaded Processor

Yoshiyasu Ogasawara, Ipppei Tate, Satoshi Watanabe, Mikiko Sato,
Koichi Sasada, Kaname Uchikura, Kazunari Asano, Mitaro Namiki and Hironori Nakajo
Tokyo University of Agriculture and Technology
2-21-16 Naka-cho, Koganei-shi, Tokyo, Japan.

Abstract

Recently reconfigurable devices such as FPGA have improved performance (gate speed and the number of gates) and reconfiguration time. Today, a reconfigurable device can integrate a large-scale processor and complex hard-wired logic. System designers found that they need a high-performance processor for their reconfigurable device based systems. To improve processor performance, a multithreaded architecture has been introduced; however, performance decreases drastically because of cache misses for shared cache among threads. Moreover, each program that a multithread processor executes may have very different cache access pattern, so that cache optimization for a multithread processor becomes much more complex compared to conventional superscalar processors. In this paper, we propose a new cache design which reconfigures cache configuration for each program on reconfigurable device. We found out optimal configuration for each program from designed cache configurations, and estimated improvement rate of reconfigurable cache. The result shows performance gains of 15.12% higher than fixed cache design.

Keywords: Reconfigurable Device, Cache Memory, Multithreaded Processor, SMT

1 Introduction

Reconfigurable devices such as an FPGA have improved spec numbers (e.g. gate speed, and gate count) drastically recently. In addition, reconfiguration time of a device has improved. Such improvement allows system designers to implement a large scale processor or a complex System-on-a-Chip (SoC) with processors in a reconfigurable device. Soft-core processors are offered by numerous companies. ARM core and MIPS cores are most widely used soft-core processor today. Also re-

cently, vendors of reconfigurable devices offer soft-core processors for SoC system designers. MicroBlaze of Xilinx [1] and Nios of Altera [2] are notable example. Although performance of these soft-core processors has improved, their performance is still far less than high performance processors such as Pentium of Intel [3]. Demand of high-performance processor soft-core for reconfigurable devices has grown recently to realize much complex SoC.

Multithread architecture is one of the prominent architectures to improve processor performance. Multithread architecture uses Thread Level Parallelism (TLP). A Simultaneous MultiThreading (SMT) processor is one of the multithread architectures. The SMT shares resources such as execution units and cache memories. It performs two or more threads simultaneously. It improves computation performance with far less hardware than a Chip Multi processors (CMP) do. Effectivity of an SMT is shown in [4]. Today, Pentium4 HyperThreading and Xeon of Intel [3] are commercially available SMT architecture processor.

SMT is very effective, however, cache contention and depletion deteriorate performance, as SMT shares caches among threads. In addition, since an SMT processor executes multiple thread programs simultaneously, memory access behavior becomes much more complicated compared to a single threaded processor. Therefore, cache memory configuration could affect the performance of SMT per each executing thread. Usually, the cache configuration of a processor is static and cannot be changed so that realizing SMT with such fixed cache configuration could deteriorate actual performance quite drastically.

In this paper, we propose a new cache design for an SMT processor in a platform of a reconfigurable device. This design aims to solve bad point on cache of multithreaded processor and to improve performance. In the generality, the cache memory

should suppress the chip area of cache on reconfigurable device because it is implemented with both many dedicated circuits and a processor on a single reconfigurable device. This design stimulates effective utilization of chip area and repression of cache capacity increase because it does not increase cache capacity, only change configuration of cache for performance gain.

Section 2 describes proposed reconfigurable cache memory. Section 3 shows performance evaluations, and we consider memory access pattern of multithreaded benchmarks in Section 4. Section 5 describes related work, and we conclude in Section 6.

2 Reconfigurable cache memory

In this section, we briefly review the background and preliminary study of our proposed cache design. Then, we introduce the outline of the reconfigurable cache memory and actual implementation.

2.1 Background of reconfigurable cache memory

Memory access pattern is dependent of run-time behavior of program and data accesses, so that the optimal cache configuration varies with programs and data. As a multithread processor executes different programs simultaneously, memory (cache) access patterns for a multithreaded processor becomes much different from a single threaded processor. And sharing cache blocks in threads occur in cache. Share cache blocks helps using cache efficiently as several threads use same cache block at regular intervals. Therefore, we consider the optimal configuration of cache that is different between single threaded processors and multithreaded processors.

We have performed a preliminary study to identify the difference between the optimal cache configurations of a single threaded processor and a multithreaded processor. We used our execution driven simulator, MUTHASI (MUlti THreaded Architecture SIMulator)[5][6] for the preliminary study. It simulates an ChiMuS (on Chip Multi SMT) PE[5][6] and provides a straightforward settings of processor parameters. We explain the parameters of MUTHASI in section 3. We executed 64x64 matrix multiplication program while changing cache parameters. However, the capacity of the cache is

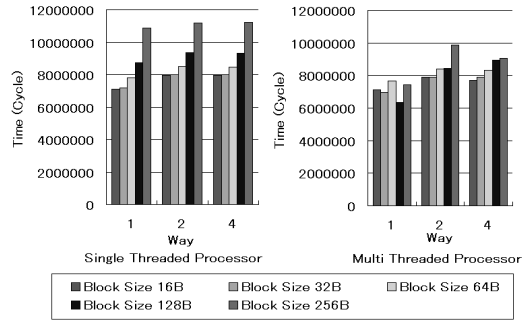


Figure 1: Performance comparison between a single threaded processor and a multithreaded processor.

fixed to 4KB. Fig.1 depicts the performance comparison.

In the figure, the horizontal and vertical axes indicate the number of ways and the number of cycles respectively. The par chart of the figure shows the block size of the cache we changed from 16 Bytes to 256 Bytes. As shown, the multithreaded processor performs better at all cache configurations. Also the graph shows that optimal cache configuration for the single threaded processor and the multithreaded processor is different. In this experimental test, we see that direct map is the best for both processors, however, the block size of 16 is best for the single threaded and 256 for the multithreaded.

Focusing on this feature of multithreaded processor, we consider performance improvement by the configuration of cache changes each program.

2.2 Outline of reconfigurable cache memory

In this subsection, we propose a **reconfigurable cache memory** that provides the optimal cache configuration for each simultaneous executing programs.

Total cache size is fixed, while the number of ways and the block size are configurable parameters for this cache. As known, number of ways increase suppresses the conflict misses, and stimulates improvement of performance. But it causes decrement of performance that access time increases. And increasing block size suppresses the compulsory misses, and stimulates improvement of performance. However cache miss penalty increases at the same time as transfer block data between cache and memory increases. Considering this trade-off, we propose the reconfigurable cache memory.

Changing tag and addition of multiplexers occur

Table 1: The Parameters of designed cache memory.

Parameter		Value
Capacity	L1-I-Cache	8KB
	L1-D-Cache	4KB, 8KB, 16KB, 32KB
	L2-Cache	512KB
Way	L1-I-Cache	1
	L1-D-Cache	1, 2, 4
	L2-Cache	.16
Block Size	L1-I-Cache	32B
	L1-D-Cache	16B, 32B, 64B, 128B, 256B
	L2-Cache	256B
Latency	L1-I-Cache	1 cycle
	L1-D-Cache	1 cycle
	L2-Cache	20 cycle

when number of ways and block size has changed, so that amount of hardware changes. But this influence is very little, because the majority of amount of hardware for cache is cache capacity [7].

Only data cache (D-cache) is reconfigured since the locality of instruction cache (I-cache) is high enough comparing with D-cache.

There are two possible schemes to reconfigure the data cache: before execution of program or between program switching. Reconfiguration is initiated by system software such as operating systems. Reconfiguration may require longer time, if we use conventional reconfigurable devices such as commercially available an FPGA. In this case, to avoid time loss, we reconfigure the cache once before executing programs. However, recently more true dynamic reconfigurable devices [8] have been available. These devices offer pseudo real time reconfiguration performance. Therefore, we think the time loss for reconfiguration will not become serious issue in the near future. In addition, if we use partial reconfiguration [9], we can change only configuration of cache, and reconfigurable time is shortened more.

2.3 Design of proposed cache memory

To evaluate the proposed reconfigurable cache memory, we have designed various reconfigurable caches shown in Table.1. The cache memory is implemented in an execution-driven simulator such as MUTHASI. The cache structure is that separated L1 instruction cache and data cache, and shared L2 instruction and data cache. LRU is adopted for cache replacement. All caches are write-back. As mentioned previously, each thread has dedicated L1 instruction cache. L1 data cache is, however, shared among all threads.

The cache capacity is fixed. Within the restric-

Table 2: The increment of lantency.

Parameter	Increment of lantency
Way 1	L1-D-Latency = 1 cycle
Way 2	L1-D-Latency + 1 cycle = 2 cycle
Way 4	L1-D-Latency + 1 cycle = 2 cycle
Block Size 16B	L2-Latency = 20 cycle
Block Size 32B	L2-Latency + 1 cycle = 21 cycle
Block Size 64B	L2-Latency + 3 cycle = 23 cycle
Block Size 128B	L2-Latency + 6 cycle = 26 cycle
Block Size 256B	L2-Latency + 12 cycle = 32 cycle

Table 3: An environment of simulation.

Parameter	Value		
PC (AT #)	1	2	4
Fetch Buffer Size	32	16	8
Dispatch Queue Size	64	32	16
Reorder Buffer Size	128	64	32
Normal Reservation Station Size	Simple ALU: 8 Complex ALU: 4		
LD/ST Reservation Station Size	8		
Branch History Table Size	1024(gshare)		
Integer ALU	Simple ALU: 4 Complex ALU: 2		
FPU	Simple ALU: 2(delay 4 cycle)		
Branch Unit	Complex ALU: 2(Mult 17 delay. Div. 30 delay)		
	1		

tion, we varied the number of ways and the block size of L1 data caches to find the optimal cache configuration for each program (Table.1). In our experiment, we also varied the cache size though it is not a part of reconfigurable parameters.

In general, an increasing number of ways brings long cache access time, and an increasing block size causes cache miss latency between L1-cache and L2-cache. We have set amount of cache access time and latency shown in Table.2 when we change number of ways and block size. The cache access time and latency are important parameter in this study, so that, we have set the cache access time and latency shown in Table.1 and Table.2 referring [7], [10], [11] and [12].

3 Performance evaluation

In this section, we show the result of our experiments. We measured the performance of the reconfigurable caches using the MUTHASI [6] and the cache memory simulator we designed. The parameter on evaluation is shown in Table.3.

We developed 3 different thread configurations for our study. 1AT (**Architecture Thread**) is a single threaded processor, 2AT and 4AT are multi-threaded processor. We run three programs; among them LU and Radix are taken from SPLASH-2 benchmark[13].

- 128 × 128 LU matrix decomposition (LU)

Table 4: Optimal configuration, cycles, speed up rate and hit rate of 4KB.

(a) LU				
	Configuration	Cycles	Speed up rate	Hit rate
1AT	Best (2way, 64B)	32005283	0.00%	96.01%
	Worst (1way, 256B)	38152700	-16.11%	90.55%
2AT	Best (4way, 128B)	21120283	51.54%	97.24%
4AT	Best (4way, 128B)	17834711	79.46%	96.58%

(b) Matrix				
	Configuration	Cycles	Speed up rate	Hit rate
1AT	Best (1way, 16B)	7097294	0.00%	53.38%
	Worst (4way, 256B)	11240356	-36.86%	55.16%
2AT	Best (1way, 128B)	6351373	11.74%	68.39%
4AT	Best (4way, 64B)	6838883	3.78%	65.49%

(c) Radix				
	Configuration	Cycles	Speed up rate	Hit rate
1AT	Best (1way, 16B)	10619505	0.00%	99.08%
	Worst (2way, 256B)	11524025	-7.85%	99.61%
2AT	Best (1way, 16B)	6351373	67.20%	98.69%
4AT	Best (4way, 64B)	5993029	77.20%	99.21%

- 64×64 Matrix multiplication (Matrix)
- Radix sort, cardinal number : 8 (Radix)

However, in this paper, due to the page restriction we omit results of Matrix and Radix. We created a parallel macro for all programs to suit multithreaded architecture.

3.1 Evaluation result

The evaluation result of LU is shown in Fig.2. As for the graph, the vertical axis of the bar graph (left side) shows number of cycles, and the vertical one of the line chart (right side) shows the hit ratio. The horizontal axis of the bar graph shows cache capacity and the number of ways. And the pattern of the bar and line shows block size of 16 Bytes ~ 256 Bytes.

LU improves its performance with increasing AT. On the whole, improvement of performance is realized by 2AT compared with 1AT. And improvement of performance is realized by 4AT compared with 2AT. In addition, if we multithread a processor, the optimal configuration of cache changes significantly.

3.2 Optimal configuration of cache memory

From the evaluation result, Table.4 summaries the optimal configuration of cache of our experiments. It shows the hit rate of the cache configurations and performance improvement, where speed up rate based on performance of 1AT (single threaded processor).

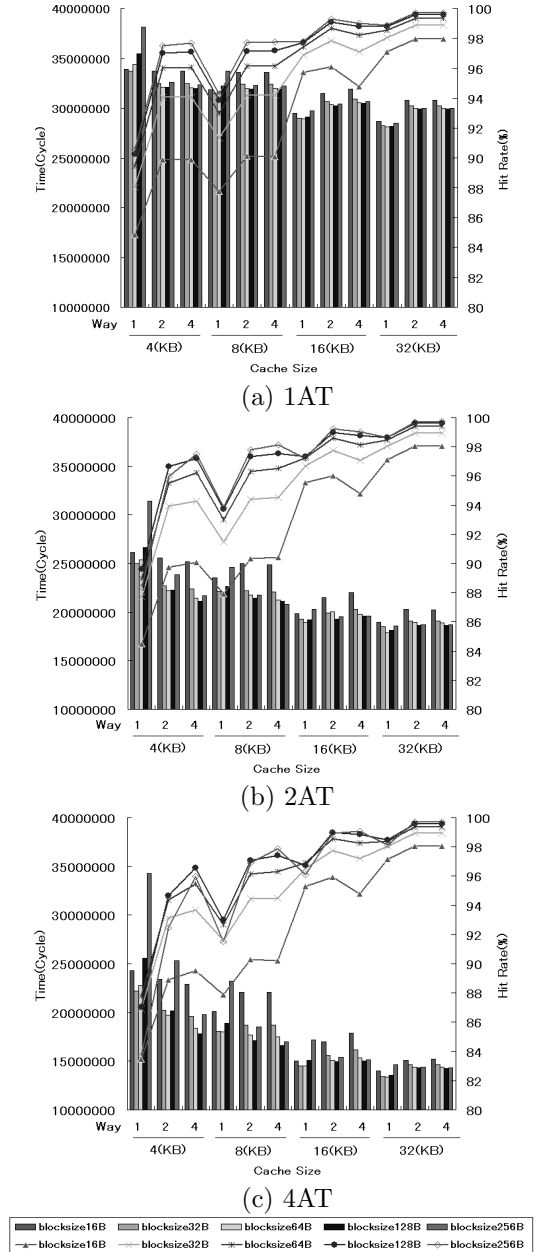


Figure 2: Evaluation result of LU.

As shown, multithreading improves performance significantly, especially for LU and Radix. To execute LU, 2AT improved performance by 51.54 %, and 4AT achieved 79.46 % improvement. On Radix, the best configuration of 2AT improves by 67.70 % comparing with 1AT, and 4AT is 77.20 % higher than 1AT.

Also shown in the table, the optimal cache configuration for multithreading becomes different from a single threaded processor. Moreover, the

Table 5: Optimal configuration of 4KB on 2AT.

Program	Optimal Configuration
LU	Way : 4, Block Size : 128B
Matrix	Way : 1, Block Size : 128B
Radix	Way : 1, Block Size : 16B

best configuration depends on the program to be executed. In our experiment, we found that for LU

a : (the number of ways = 2, the number of block size = 64Bytes) for 1AT,

b : (the number of ways = 4, the number of block size = 128Bytes) for 2AT.

derive the best performance. For Radix experiment, we found

a :(the number of ways = 1, the number of block size = 16Bytes) for 2AT,

b :(the number of ways = 4, the number of block size = 64Bytes) for 4AT.

are the best cache parameters.

Based on Table.4, we show the optimal configuration of 2AT for each benchmarks in Table.5. We can state that the cache configuration affects the performance of multithreaded processor significantly. To reconfigure the data cache, we can achieve much better performance while fixing the cache size and the number of threads. The optimal configuration in LU and Matrix is 128 Byte in block size. But, this configuration causes performance decrement in non-continuous access, which does not employ in general cache of static device. However, these particular kind of configuration is effectively in the specific program. Thus the effectivity of the reconfigurable cache memory which can actualize the particular configuration is much higher.

3.3 Performance improvement of reconfigurable cache memory

This subsection shows the performance improvement rate of reconfigurable cache memory. For example, we consider executing LU, Matrix and Radix continuously in 4KB in cache capacity and 2AT. Here, we have named four configuration of cache (A), (B), (C) and (RECONF) respectively.

(A) : The configuration is fixed with 4-way/128Bytes

(B) : The configuration is fixed with 1-way/128Bytes

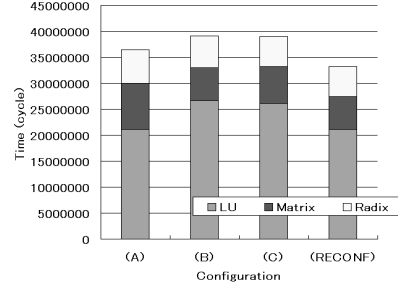


Figure 3: Performance comparison of cache configurations.

Table 6: Improvement of reconfigurable cache memory.

The Object of Comparison	The Rate of Improvement
(A)	8.84%
(B)	15.12%
(C)	14.85%

(C) : The configuration is fixed with 1-way/16Bytes

(RECONF) : The reconfigurable cache memory

The performance comparison of cache configurations is shown in Fig.3. And the rate of improvement of reconfigurable cache memory is shown in Table.6.

From Fig.3, (RECONF) improves the performance compared with the fixed configuration. And from Table.6, the rate of improvement is 15.12 % compared with (B).

We have studied the performance comparison in three benchmarks and four configurations of cache. However the reconfigurable cache memory is effective in another condition. Especially, we expect the reconfigurable cache memory can gain much higher performance when programs needing particular configuration are executed.

4 Memory access pattern of multithreaded benchmarks

In this section, we analyze the algorithms further to assess how multithreaded approach improves performance, and the cache optimization contributes performance improvement.

4.1 LU matrix decomposition

LU from the SPLASH2 divides matrix into small block of $B \times B$ (shaded area on Fig.4) and calculates

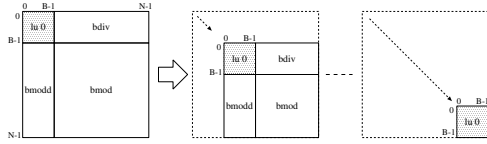


Figure 4: Overview of LU processing.

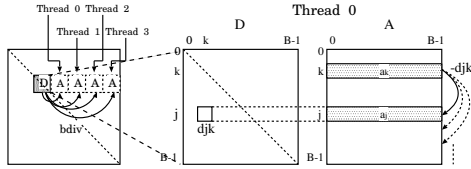


Figure 5: Overview of bdiv processing.

by this small block. The processing of LU is divided into four processing of lu0, bdiv, bmodd and bmod such as Fig.4. Due to space limitation, we explain only the bdiv.

The bdiv performs vector subtraction of k-th row from j-th row as $a_j - d_{j,k} a_k$ such as Fig.5 ($j < k$), where the block of processing object is matrix A ($a_{y,x}$), and the block of diagonal is matrix D ($d_{y,x}$).

By multithreading, the bdiv allocates a thread into the matrix A of subject to processing object, to do thread parallelization. And by the subtraction of vectors, the memory access works on the direction of rows.

In LU, the spatial locality is high because the access of rows direction is requested. By multithreading, it is parallelized, and the spatial locality becomes much higher. Therefore, block size increase achieves hit rate improvement. When cache capacity is small, index conflict happens, and the hit rate is improved by the increasing number of ways.

4.2 Matrix multiplication

The Matrix is a benchmark programmed by the authors. In thread parallelization, the matrix of this study divides the column of matrix B into M equal parts, where M is number of threads doing parallelization. Memory access pattern is shown in Fig.6. The matrix A accesses to row direction, and the matrix B accesses to column direction.

In this case, the spatial locality is high, because the row access is linear. By multithreading, the spatial locality becomes higher because multiple threads use one row. The hit rate is improved by the increasing block size such as LU.

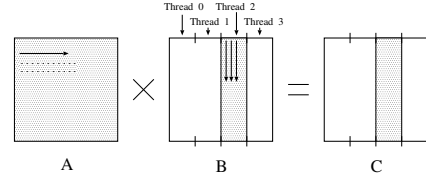


Figure 6: Memory access pattern of Matrix.

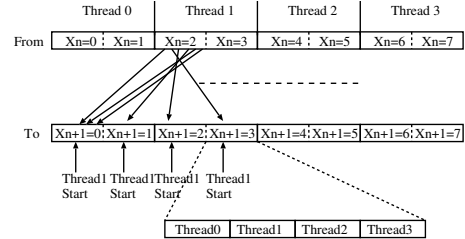


Figure 7: Memory access pattern of Radix.

4.3 Radix sort

The main processings of Radix are data sort and data access.

The main processings of Radix are data sort and data access. The array "From" in Fig.7 had sorted until n-th digit in octal notation. Fig.7 shows the memory access pattern to store the data, sorted by n+1-th digit, in the array "To." And each zone in the array "To" is further divided into ATs.

The Radix accesses linearly in each zone of the data array "From." But the working-set of Radix is diffusive because the distributed access in the data array "To." Therefore, the locality of memory access pattern is small, and the area of the data array increases by large cardinal number. The memory access pattern of Radix is similar to random access, so that, Radix is improved the performance by multithreading, but the optimal configuration of cache does not change compared with single-thread processor.

5 Related work

In this section, we highlight related works in our study.

Ranganathan et al have proposed the design of dividing cache in [11]. This design divides a set associative cache into several caches as usage. Ranganathan have employed this design for a media processing, and Tanaka have applied this design to the multi-threaded processor [14]. Suh et al who

has focused on the temporal locality of a thread in a multi threaded processor, divides L2-cache into each thread [12]. These studies use a fundamental configuration of cache, and divide cache. Therefore, they propose designs of dividing cache rather than reconfiguring cache.

Kim et al has proposed the design which reconfigures cache to dedicated circuit using reconfigurable device as with our study [15]. Kim has focused on an unallocated portion of a cache in a media processor. This design releases an unallocated portion of a cache on an FPGA, and reconfigures its LUT (Look Up Table) to dedicated circuit such as FIR filter. In contrast, the reconfigurable cache memory of our study is the design which focuses on cache only, so that, the reconfiguration becomes simpler, and general versatility is higher. And this design stimulates effective for utilization of chip area and repression of cache capacity increase because it does not increase cache capacity, only changes number of ways and block size for performance gain.

6 Conclusions

In this paper, we proposed a new cache design to optimize cache performance for a multithreaded processor. As we examined in section 3 and 4, cache access pattern varies for each program and data. Under the constraint of the cache capacity of a processor, to adapt the parameters of a cache memory, the number of ways, and the size of block, we found that we can achieve more than 15.12 % performance gain.

In this paper, we found the optimal configuration of cache by the evaluation. When it's used for real, there are two presumable ways to find out optimal configuration. The one hand, the way to find by statistics with data of cache performance. Or the other hand, the way to find by scheduling of cache configuration by admission test.

Although the evaluation presented with OChiMuS PE in this paper, the proposals made in this research are not limited to this architecture. For example, the reconfigurable cache memory is applicable easily on commercial multi-threaded processor implemented on a reconfigurable device. And the optimal configuration of cache is also changed by a program on 1AT (single threaded processor) in the evaluation. Therefore, we consider our cache design is available not only on multithreaded processor but also single threaded processor.

We plan to estimate strategy of block replacement as configuration parameter. After that, we will develop a reconfigurable cache memory, and will consider a new model of reconfigurable cache memory.

References

- [1] MicroBlaze : http://www.xilinx.com/ipcenter/catalog/logicore/docs/microblaze_risc_32bit_proc_final.pdf
- [2] Nios 3.0 CPU : http://www.altera.co.jp/literature/ds/ds_nios_cpu.pdf
- [3] D. Marr, F. Binns, D. Hill, G. Hinton, D. Koufaty, J. Miller, M. Upton : "Hyper-Threading Technology Architecture and Microarchitecture" , *Intel Technology Journal*, Vol.6, pp.4-15, 2002.
- [4] D. Tullsen, S. Eggers, H. Levy : "Simultaneous Multi-threading:Maximizing On-Chip Parallelism", *Proceedings of 22d Annual International Symposium on Computer Architecture*, pp.392-403, 1995.
- [5] Y. Ogasawara, N. Kato, M. Yamato, M. Sato, K. Sasada, K. Uchikura, M. Namiki and H. Nakajo : "A New Model of Reconfigurable Cache for an SMT Processor and its FPGA Implementation", *Proceedings of the 2005 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'05)*, Vol.II, pp.447-453, 2005.
- [6] K. Sasada, M. Sato, S. Kawahara, N. Kato, M. Yamato, H. Nakajo, M. Namiki : "Implementation and Evaluation of a Thread Library for Multithreaded Architecture" , *PDPTA'03*, pp609-615, 2003.
- [7] J L.heneesy, D A.Patterson : "Computer Architecture A Quantitative Approach Third Edition", *Morgan Kaufmann Publishers*, 2002.
- [8] Microprocessor Report : <http://www.ednjournal.com/content/issue/2004/05/micro/micro.html>
- [9] Partial Reconfigurability : http://www.xilinx.co.jp/products/design_resources/design_tool/grouping/adv_design_tech.htm
- [10] IA-32 Intel Architecture Software Developer's Manuals : http://developer.intel.com/design/pentium4/manuals/index_new.htm
- [11] P. Ranganathan, S. Adve, N. P. Jouppi : "Reconfigurable Caches and their Application to Media Processing", *ISCA-27*, pp.214-224, 2000.
- [12] G.E.Suh, L.Rudolph, S.Devadas : "Dynamic Partitioning of Shared Cache Memory" , *The Journal of Supercomputing Architecture*, pp.7-26 , 2004.
- [13] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, A. Gupta : "The SPLASH-2 Programs: Characterization and Methodological Considerations", *In Proceedings of the 22nd International Symposium on Computer Architecture*, pp.24-36, 1995.
- [14] K. Tanaka : "Fast Context Switching by Hierarchical Task Allocation and Reconfigurable Cache", *IWIA2004*, pp.20-29, 2003.
- [15] Hue-Sung Kim, Arun K. Somani, Akhilesh Tyagi : "A Reconfigurable Multi-function Computing Cache Architecture", *In Proceedings of the 2000 ACM/SIGDA 8th International Symposium on Field Programmable Gate Arrays*, pp85-94, 2000.