

# Basic Mechanisms of Thread Control for On-Chip-Memory Multi-threading Processor

Takanori MATSUZAKI †, Hiroshi TOMIYASU ‡, Makoto AMAMIYA †

† Graduate School of Information Science and Electrical Engineering,  
Kyushu University

6-1, Kasuga-Koen, Kasuga, Fukuoka, 816-8580, Japan  
{takanori, amamiya}@al.is.kyushu-u.ac.jp

‡ Institute of Information Sciences and Electronics,  
University of Tsukuba

1-1-1 Tennodai, Tsukuba, Ibaraki, 305, Japan  
tomiyasu@is.tsukuba.ac.jp

## Abstract

In this paper, we describe basic mechanisms of thread control for the FUCE processor. FUCE means FUSion of Communication and Execution. The goal of the FUCE processor project is to fuse the intra-processor execution and inter-processor communication. In order to achieve this goal, the FUCE processor integrates the processor units, memory unit and communication units into a single chip. The FUCE processor has mechanisms for pre-loading thread context and hiding memory access latency. With these mechanisms, no data cache memory is required, since memory access latency can be hidden due to a simultaneous multi-threading mechanism and the on-chip-memory system with broad-bandwidth low latency internal bus of the FUCE processor. This approach can reduce the performance gap between instruction execution, and memory and network accesses.

**Keywords** Multi-threading, pre-loading thread context, hiding of memory access latency, on-chip-memory processor, on-chip multi-processor.

## 1 Introduction

Currently, communication and VLSI device technologies are advancing towards higher and higher speeds and are becoming larger in scale. For example, optical-fiber transmission-line technology is now achieving Giga-bits/sec speeds and will achieve Tera-bits/sec speeds in the near future. New communication protocols, e.g., IP on WDM and IP on SONET, are now under development. In addition, hardware VLSI device technology is advancing to larger scale integration VLSI's with Giga-gate logic and Giga-bit memory on chip, and higher clock speeds of several Giga-Hz.

Software technologies including processor architectures, in contrast, are still developing within the conventional framework. The development of new architecture and software technologies is urgently required.

Against the background of these hardware and software technology trends, we are pursuing the FUCE project at Kyushu University. FUCE means FUSion of Communication and Execution. The main objective of this research is, as the name shows, to develop a new architecture that truly fuses communication and computation. The FUCE project aims to develop a new processor-architecture and kernel-software (operating system) for fusing computation and communications. We call the processor the FUCE processor, and the kernel-software CEFOS (Communication and Execution Fusion OS) [2]

The FUCE processor is an on-chip-memory processor developed based on a fine-grain multi-threading concept. In the FUCE processor, a thread is a tiny process executed without preemption. The fine-grain multi-threading technique promises high performance in fusing communication and internal execution. Both event handling, i.e., incoming/outgoing messages and I/O, and internal process execution are controlled by a uniform thread execution mechanism. The on-chip-memory processor technique also promises high performance in hiding memory access latency. No data cache memory is required in the FUCE processor, since the on-chip-memory system provides low latency memory accesses.

This paper introduces the FUCE processor and discusses the simultaneous multi-threaded execution mechanism and on-chip-memory system. Section 2 presents an overview of the FUCE processor. Section 3 discusses the FUCE process and FUCE threads. Section 4 covers the effect of hiding the memory access latency. Section 5 discusses the originality of the FUCE processor in comparison with related work.

## 2 Overview of the FUCE processor

The objective in designing the FUCE processor is to fuse the intra-processor execution and inter-processor communication so that the mechanism reduces the performance gap between intra-processor execution and inter-processor communication by integrating into one chip the execution units, communication units and memory unit.

An overview of FUCE Processor is shown in Figure 1.

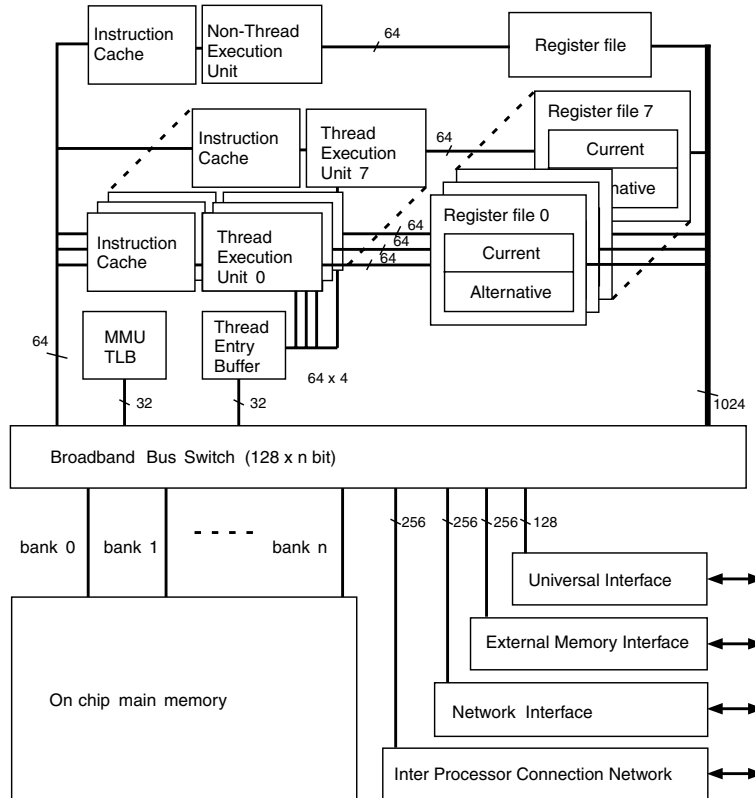


Figure 1: the FUCE Processor

The FUCE processor executes multiple threads in parallel and concurrently. It is designed as a multi-processor on a single chip to support high-speed multi-thread execution. In the on-chip multi-processor, multiple execution units, a ready thread queue control and event-handling units are incorporated into one chip. In addition to multiple execution units, a communication control unit and a main memory are also integrated into the chip. Another important design issue is the data path between execution units and memory units. The FUCE processor incorporates a broad-bandwidth low-latency internal bus. It also has external memory units (e.g. off-chip-memory or disks), which are connected to an external memory interface. It uses on-chip-memory as main memory and uses the external memory units as secondary memory.

Our design specifications for the FUCE processor are as follows:

1. Multiple units for thread execution.
2. Highly efficient thread execution.

3. Large size of register files.
4. High-speed broadband bus.
5. Large on-chip-memory.

The FUCE processor is being designed with future VLSI technology in mind. Table 1 shows the specifications of the FUCE processor. In the near future, VLSI technology will be able to achieve 800 Mega-transistors/chip with a chip size of  $600mm^2$ [11]. The FUCE processor uses half the area of the chip for on-chip memory. Also, we estimate that the number of transistors for a thread execution unit will be up to 5 Mega-transistors/unit, so 8-16 thread execution units and the non-thread execution unit will require 100 Mega-transistors. Therefore, the FUCE processor can integrate other units (e.g. communication control units and a broadband internal bus) into one chip.

Table 1: Specification of On-chip FUCE Processor

	2005	2010
Clock cycles	4 Giga-Hz	10 Giga-Hz
On-chip-memory Capacity	256 Mega-Bytes	1 Giga-Byte
On-chip-memory Speed	2 Giga-Hz	5 Giga-Hz
Internal Bus Speed	512 Giga-Bytes/sec	2.5 Tera-Bytes/sec

## 2.1 Execution Unit

The FUCE processor has two kinds of instruction execution units. One is the thread execution unit, and the other is the non-thread execution unit. The thread execution units execute threads without preemption. The non-thread execution unit executes non-thread code, which handles interrupts or exceptions for the OS kernel. Thread execution units have no preemption mechanism, while the non-thread execution unit can suspend and resume operations.

The FUCE processor has multiple thread execution units, each of which executes a thread independently. We believe that the memory system performance is more important than complicated high performance execution such as speculative execution and out-of-order execution. Therefore, the thread execution units of the FUCE processor are constructed with a simple pipeline structure. Here, the thread is defined as a sequence of instructions that is executed exclusively without any interruption except for some emergency cases such as infinite loops.

The features of the thread execution units are:

1. Thread execution units issue two instructions simultaneously and in order, and execute those two instructions in parallel.
2. Thread execution units transfer a set of registers in one instruction. We call this mechanism **block load/store**.
3. The execution pipeline of a thread execution unit will not stall while loading data from memory. We call this mechanism **non-blocking load**.
4. Thread execution units have two sets of register files: a current register file and alternative register file. The current register file is used for the thread execution in the foreground. The context of a ready thread is pre-loaded into the alternative register file in the background. The alternative register file takes the place of the current register file when the next ready thread runs.
5. Thread execution units have a thread entry buffer. This buffer is a hardware queue, which holds ready threads.

## 2.2 On-Chip-Memory

Current processors, which have off-chip memory, suffer from processor-pin bottleneck. It is therefore difficult to expand the memory buses of current processors. On the other hand, on-chip-memory processors scarcely suffer from processor-pin bottleneck, because on-chip-memory processors do not require

processor pins for the memory buses and it is easy to expand their memory buses. Furthermore, on-chip-memory processors make low-latency memory access possible. Low latency (e.g. around 4-8 cycles) and broadband (e.g. 512 Giga-Bytes/sec) data transfer are two features of the FUCE processor memory system.

The FUCE processor can also hide the memory access latency. In order to hide the memory access latency, it implements mechanisms that allow the thread execution units to access the on-chip-memory with low latency. These mechanisms are the block load/store and the non-blocking load. It can also reduce the overhead of memory access involved in switching the thread context thanks to the pre-loading of thread context.

The FUCE processor assumes that thread instructions are re-ordered by the compiler to pre-fetch the data. Re-ordering of instructions is such that the thread execution unit pre-fetches the data into a register during the thread execution. This pre-fetch instruction uses the non-blocking load and block load/store. This pre-fetch mechanism will avoid pipeline stall and reduce the idle time while the data are loaded into the register. In addition, the block load/store reduces the number of data transfer instructions.

With the memory units integrated into the chip, memory access becomes possible in four to eight cycles, and high-speed data transfer is performed between the execution unit and the memory unit. In addition, the FUCE processor has a mechanism for hiding the memory access latency, i.e. pre-loading of thread context. Note here that, even though the FUCE processor hides a low memory access latency (e.g. 4-8 cycles), it can not hide a large memory access latency of 100 cycles or more. Since the memory access to off-chip-memory takes too many cycles for the memory access latency to be hidden, the FUCE processor provides only on-chip-memory.

In fine-grain multi-threading, threads consist of small chunks of instructions and thread execution will terminate before data in cache memory are used more than once. In this situation, the gains afforded by cache memory are less than the overhead necessary for the cache memory control. But the FUCE processor can hide the memory access latency, and therefore no data cache memory is required for fine-grain multi-threading. On the other hand, the FUCE processor employs an instruction cache due to the principle of locality of instruction sequence in a thread.

In the FUCE processor memory system design, internal data transfer is more important than the data transfer between internal main memory and external memory.

### 3 FUCE process and FUCE threads

The basic structure of process and threads controlled in the FUCE processor is shown in Figure 2. The FUCE processor's process has more than one thread. The process is a unit of resource operation and the thread is a unit of processor assignment. Threads in the same process share elements of their processing environment such as a stack and a virtual memory space.

The basic features of the FUCE thread are as follows:

1. The FUCE thread is a fine-grain multi-thread. A large amount of the FUCE thread is assumed to run concurrently. Concurrent FUCE thread executions hide the communication latency.
2. The FUCE thread is a tiny process with no interruption. Therefore, the FUCE thread never suspends until it encounters its thread termination instruction. Furthermore, it has no limits in principle on its execution time.
3. The FUCE thread is a lightweight thread, and the lightweight thread can reduce the thread switching overhead.
4. The FUCE thread never accesses Off-Chip memory while it is running. Therefore, the FUCE thread is split when its execution has a large latency because it must access off-chip memory.

The FUCE processor can execute multiple threads belonging to different processes, enabling it to execute multi-threaded code on the multiple thread execution units. This approach can obtain sufficient ready threads to keep all of the thread execution units busy. If the FUCE thread has off-chip memory access, such as accessing external memory or accessing another node processor's memory, it will be separated into two threads, which are the caller thread and the recipient thread. In this way, the FUCE processor will be able to hide the latency of accessing off-chip memory.

### 3.1 Basic Mechanisms of Thread Control

The basic mechanisms of thread control are shown in Figure 3. To simplify the hardware mechanism of thread control, an operating system controls the thread execution in the FUCE processor. The thread execution control is performed through the software and hardware queue.

The operating system has one software queue, and controls the thread execution with the software queue. The software queue has threads, which are synchronized threads and can be executed immediately. Also, the software queue is able to have many threads which belong to another process. The operating system manages many processes, and each process has one thread scheduler.

The operating system controls the thread execution in the FUCE processor, synchronizing the threads. In order to synchronize the threads, the operating system has thread schedulers, which synchronize the threads in each process. The thread scheduler controls the thread execution order dynamically by synchronizing threads, and enqueueing synchronized threads in the software queue.

The hardware queue consists of the thread entry buffer. A Ready thread is registered in the thread entry buffer, and waits to be assigned to an idle thread execution unit. The operating system, when notified that the thread entry buffer is empty, moves a thread to the thread entry buffer. This notification is managed by hardware, but the transaction is controlled by software. The FUCE processor also has special instructions which control thread registration.

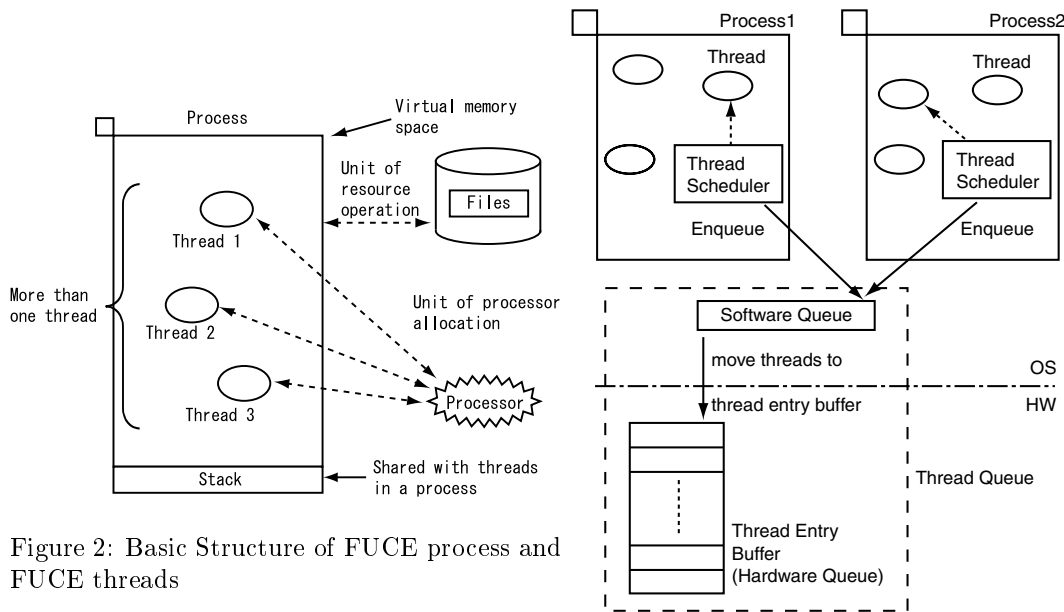


Figure 2: Basic Structure of FUCE process and FUCE threads

Figure 3: Basic Mechanisms of Thread Control

## 4 Hardware Support Mechanisms for Thread Execution

### 4.1 Thread Context Pre-loading

Context switches occur quite often in fine-grain multi-threading, and the overhead of a context switch is quite serious. In order to reduce this overhead, the FUCE processor pre-loads the thread context. In pre-loading, the processor uses two sets of register files and the thread entry buffer. The pre-loading mechanism allows the FUCE processor to achieve fast thread-context switches.

The FUCE Processor pre-loads the next thread context using the pre-loading unit, which is a special unit for the thread pre-loading. The FUCE processor has multiple thread pre-load units, each of which is connected to a thread execution unit. The pre-load units pre-load the next thread context with reference to the data included in the header of the next thread instructions. Figure 4 shows an overview of thread context pre-loading. The basic mechanism and function of thread context pre-loading are as follows:

1. A ready thread is assigned to an idle thread execution unit. The allocated thread uses the current register file when it begins to run.

2. When the thread execution unit begins to run the allocated thread, a new ready thread in the thread entry buffer is assigned to the thread execution unit.
3. While the thread execution unit is executing the allocated thread in foreground, it loads the thread context of a new ready thread, which is to be executed right after the current thread execution terminates, into its alternative register file in the background.
4. After the thread execution terminates, the thread execution unit exchanges the alternative register file and the current register file, and begins to execute the new thread.

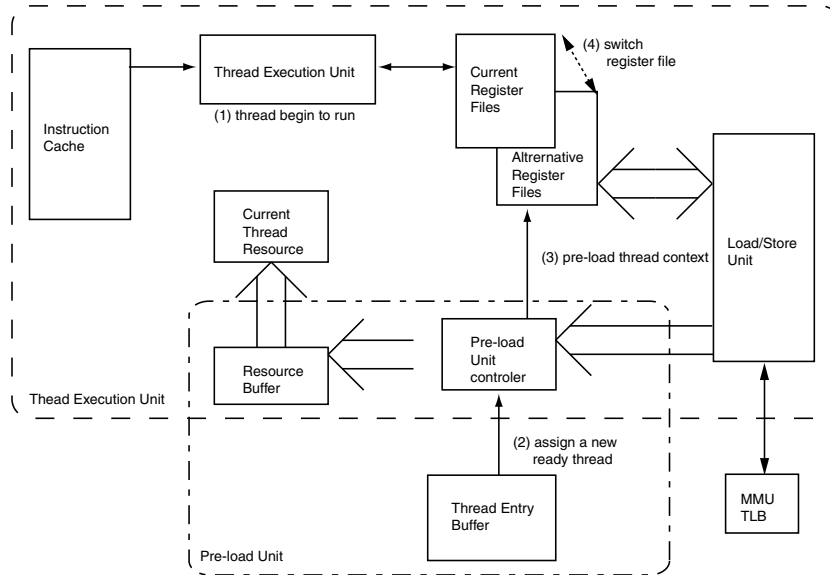


Figure 4: Pre-loading of Thread Context

## 4.2 Effect of thread context pre-loading

We evaluated the effect of the thread context pre-loading, described in section 4.1 using two benchmark programs:

- **Matrix:** 1000 × 1000 matrix multiplication.
- **8-Queens:** Finding all solutions of the 8-queens problem.

The specifications for the thread execution units used in the evaluation are shown in Table 2. In the evaluation, we used 8 thread execution units.

Table 2: Specifications of the thread execution units

Instruction Issuing Rate	2 instructions/clock-cycle
Number of Registers	64 × 2
Block-data per Transfer	4 blocks/instruction
Memory Access Latency	4, 6, 8, 10 cycles
Latency of Floating Point Execution	4 cycles

### 4.2.1 Matrix

The effect is examined for the matrix multiplication of 1000 × 1000 matrices. In this evaluation, we evaluated two cases of execution time and pipeline stall, and examined them for memory access latencies of 4, 6, 8 and 10 cycles: (a) With thread context pre-loading, (b) Without thread context pre-loading. Figure 5 shows the execution time and pipeline stall time.

In the Matrix problems, most of the instructions are load instructions, and the other instructions are multiplication and addition. The FUCE processor can load the data into the register without pipeline stall by using thread context pre-loading. So it can execute matrix problems with only a tiny pipeline stall (about 0.5%) (Figure 5: (a) With thread context pre-loading). On the other hand, It can not hide pipeline stall, when it does not use thread context pre-loading (Figure 5: (b) Without thread context pre-loading).

We can see from these results that thread context pre-loading makes a clear contribution to reducing pipeline stall and increasing the performance of thread execution.

#### 4.2.2 8-Queens

We evaluated the execution time and the pipeline stall time in finding all solutions of the 8-queens problem. In this evaluation, on-chip memory access latency is 4 cycles. Figure 6 shows the execution time and the pipeline stall time.

In the 8-queens problem, the load instructions and the store instructions constitute about 35 percent of all the instructions. In addition, the load instruction and the store instructions are distributed. So, it is difficult for the FUCE processor to hide memory access latency (Figure 6: (a) Normal instructions). However, the FUCE processor can reduce on-chip memory access latency by separating instructions into threads and re-ordering thread instructions (Figure 6: (b) Re-ordering Threaded instructions). These approaches use thread context pre-loading.

We can see from these results that thread context pre-loading can reduce the execution time and pipeline stall time. This is because thread context pre-loading reduces the overhead of thread context switching.

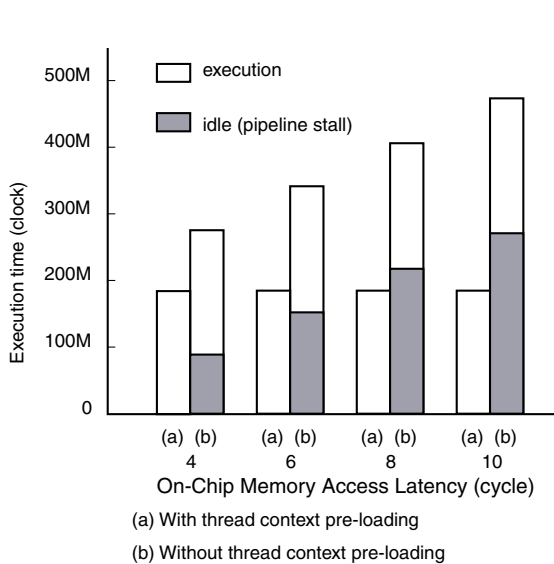


Figure 5: Matrix Multiplication

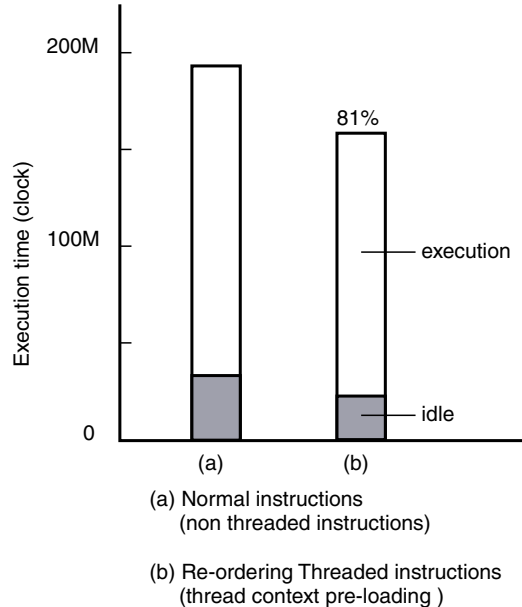


Figure 6: 8-Queens Problem

## 5 Related Work

The importance and feasibility of the fine-grain multi-threading technique are generating interest in the parallel computing research field. For instance, the MTA machine has been commercialized[3] and the HTMT project is enthusiastically proceeding towards the goal of a Petaflops machine[4]. The concepts behind the FUCE machine have emerged from our research on multi-threading architecture and the parallel processing language V[5][6].

Because on-chip-memory processors achieve low latency and high-speed memory access, the on-chip-memory processor technique is being researched in the field of high performance computer systems. For

instance, PPRAM is an architectural framework for merged memory/logic ASSPs (Application-Specific Standard Products)[7]. Hydra[8] integrates multiple processors and cache memory on a single chip. The FUCE processor integrates not only multiple execution units and main memory but also communication control units into a single chip, in order to fuse communication and internal execution.

A great deal of research into on-chip multi-processors is also currently being pursued around the world. MP98[9], for example, supports efficient thread creation and execution through mechanisms involving inheritance of register values, resolution of data dependencies and speculative execution. However, these mechanisms make the hardware logic more complicated. The FUCE processor does not rely on such complicated mechanisms. SMT[10] is a simultaneous multi-threading processor, in which multiple threads are dynamically assigned to execution units at both the instruction level and the thread level. The FUCE processor is being developed based on the technique of fine-grain multi-threading and runs multiple threads concurrently in the multiple thread execution units. The FUCE processor is similar to SMT in that both rely on concurrent multi-thread execution. However, the multi-threading approach of the FUCE processor, in sharp contrast to that of SMT, is to assign each thread to a single execution unit and execute it exclusively on that execution unit in order to make the hardware logic simple and transparent.

## 6 Conclusion

This paper has discussed the basic mechanisms of thread control for on-chip-memory multi-threading processor architecture, and evaluated the effect of thread context pre-loading in the FUCE processor. In the FUCE processor architecture, we are making use of the technique of on-chip-memory processing. In addition, we are using thread execution support mechanisms, such as thread context pre-loading and the hiding of memory access latency. These mechanisms are very effective for thread execution.

Through these approaches, the FUCE processor can reduce the performance gap between instruction execution, and memory and network access. The architecture of the FUCE processor, which integrates on-chip-memory VLSI processor construction and simultaneous multi-thread processing, will provide solutions to important future technical issues in high performance parallel and distributed processing.

## Acknowledgments

This research was pursued with support for R&D activities in the info-communications area from the Telecommunication Advancement Organization of Japan.

## References

- [1] T. Matsuzaki, et al. An Architecture of On-Chip-Memory Multi-threading Processor, Proc. of International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems (IWIA01), 1 2001, To Appear.
- [2] M. Amamiya, et al. An architecture of fusing communication and execution for global distributed processing, In SSGRR2000 Computer and Business Conference, 8 2000.
- [3] G. Alverson, et al. Tera Hardware-Software Cooperation, Proc. Supercomputing, San Jose, 1997.
- [4] G. Gao, et al. Hybrid Technology Multithread Architecture, Proceedings of The Sixth Symposium on The Frontiers of Massively Parallel Computation (Frontiers '96), pp.98-105, October, 1996
- [5] M. Amamiya, et al. Datarol: A Parallel Machine Architecture for Fine-Grain Multithreading, Proc. Third Working Conference on Massively Parallel Programming Models, London, 1997.
- [6] M. Amamiya, et al. Co-Processor Design for Fine-grain Message Handling in KUMP/D, Proc. European Conference on Parallel Processing, Passau, pp.779-788, 1997.
- [7] K. Murakami, et al. Parallel Processing RAM (PPRAM), Japan-Germany Forum on Information Technology, Nov. 1997.
- [8] L. Hammond, et al. "The Stanford Hydra CMP", IEEE Micro, Vol. 20, No. 2, March/April 2000
- [9] N. Nishi, et al. A 1GIPS 1W Single-Chip Tightly-Coupled Four-Way Multiprocessor with Architecture Support for Multiple Control Flow Execution, In Proc. ISSCC2000, WP25.5.
- [10] Dean M. Tullsen et al. "Simultaneous Multithreading: Maximizing On-Chip Parallelism," Proceedings of the 22nd Annual International Symposium on Computer Architecture, June, 1995.
- [11] The International Technology Roadmap for Semiconductors (ITRS), <http://public.itrs.net/>
- [12] Mosys, Inc. <http://www.mosysinc.com/mhhome/>