Near fine grain parallel processing using a multiprocessor with MAPLE

T. ABE † K. IWAI †† T. MORIMURA † R. OGAWA† K. YASUFUKU†
H. AMANO†
† Department of Computer Science, Keio University 3-14-1, Hiyoshi Yokohama, 223-8522, Japan.
asca@am.ics.keio.ac.jp
†† National Defence Academy,
1-10-20 Hashirimizu Yokosuka, 239-8686 , Japan,
iwai@nda.ac.jp

Abstract

Multi-grain parallelizing scheme is one of effective parallelizing schemes which exploits various level parallelism: coarse-grain(macro-dataflow), medium-grain(loop level parallelizing) and near-fine-grain(statements parallelizing) from a sequential program. A multi-processor ASCA is designed for efficient execution of multi-grain parallelizing program.

A processing element called MAPLE are mainly designed for near-fine-grain parallelism, and has two modules called MAPLE core and DTC. The MAPLE core is a simple RISC processor which executes every operation in a fixed time and realize direct register to register transfer. The DTC realize a software controlled cache by instructions which are generated by the compiler. With a static scheduling, near-fine-grain parallel processing is efficiently performed using a communication mechanism with receive registers, and non-synchronization operation mechanism.

Through implementation of the prototype chip and clock level simulation, it appears that the performance of a single chip multi-processor with 4 MAPLEs is close to those of modern super-scaler processors in spite of small hardware and low clock frequency.

KEYWORDS: processor, cache, static scheduling, multi-grain parallelism, parallel computing system

1 Introduction

Automatic parallelizing compilation schemes are important for common programmers to save their efforts for writing the effective parallelizing code for the target machine. Although these schemes are useful for various multi-processors, maximum performance will be obtained with a multi-processor architecture whose processor, memory system, and interconnection network are tailored for the schemes. For this purpose, we have proposed a multi-processor system ASCA (Advanced Scheduling oriented Computer Architecture). ASCA system is designed for the multi-grain parallelizing scheme [1], one of effective parallelizing schemes.

This scheme exploits parallelism from a sequential program in various levels: coarse-grain parallelism(macro-dataflow computation) [2], medium-grain parallelism(loop level parallelism) and near-fine-grain parallelism(statement level parallelism) [3]. The former two types of parallelism mainly concern with an interconnection called R-Clos and total system of ASCA, while the processor core MAPLE and dedicated cache called DTC are designed for the latter near-fine-grain parallelism.

Here, a processing element of ASCA which consists of two chips: MAPLE processor core and Data Transfer Controller is designed and implemented. Near-fine-grain parallel execution using multiple processing elements is evaluated.

2 ASCA multi-processor

2.1 Multi-grain parallel processing

A common compiler focuses on loop structures in a program, and detects parallelism between iterations. It is called loop-level parallelism or medium-grain parallelism. Although this type of parallelism is efficient in a class of scientific programs, almost no performance enhancement is expected for programs including complicated loop structures.

For such programs, a coarse-grain parallel processing[2] which uses a parallelism between large modules of programs corresponding to loops themselves and subroutines is often effective. A near-fine-grain parallel processing[3], which uses a statement level parallelism in a program module, is also effective to make the best use of inherent parallelism. Multi-grain parallel processing[1] is a scheme which exploits above three levels: coarse-grain, medium-grain and near-fine grain.

Although this scheme is applicable to any parallel machines, architectural supports are required. For example, a large program module called macro task must be assigned dynamically according to macro data flow graph in a coarse-grain parallel processing. On the other hand, high speed and light weight communication and synchronization between processing elements are required for efficient near-fine-grain parallel processing. In this level, if communication between processing element is completely scheduled at the compile time, all synchronization codes can be e-

liminated. For the non-synchronized execution mode, the computation time and communication time of each hardware block must be done in a constant time which can be treated in the compiler. Thus, a dedicated multi-processor architecture is required for efficient execution of multi-grain parallel processing.

2.2 ASCA Multiprocessor

ASCA multi-processor[4] has been developed as a project supported by STARC to establish architectural techniques in a dedicated architecture for multi-grain parallel processing.

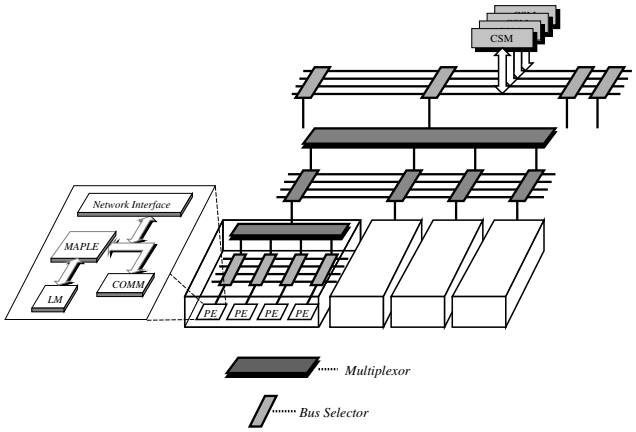

Figure 1: Structure of ASCA

As shown in Figure 1, ASCA multi-processor consists of multiple clusters which are connected with R-Clos interconnection network[7][8]. A node of macro-data flow graph which exploits coarse-grain parallelism is assigned into each cluster. Each cluster is a multi-processor system consisting of dedicated processor core called MAPLE and specialized cache which are designed so as to make the best use of near-fine-grain parallelism as well as traditional loop-level parallelism. Although a cluster will be implemented as a single chip-multi-processor with the near future technology, a processing element is implemented on a board in the first prototype of ASCA. Now, three chips for key components of the board are available[9], and a board including a processing element is now under development.

In this paper, we focus on a near-fine-grain parallel processing in an ASCA cluster consisting of MAPLE processors and dedicated cache systems. Other techniques investigated in ASCA project are shown in our previous papers[4][5][7][8][9].

3 Processing element MAPLE

3.1 Outline of MAPLE

MAPLE(Multiprocessor system ASCA Processing eLEment) is a processing element in ASCA developed for near-fine-grain parallel processing. As shown in Figure2, it consists of a processor, Local Memory(LM), Communication Memory(CM) and Network Interface.

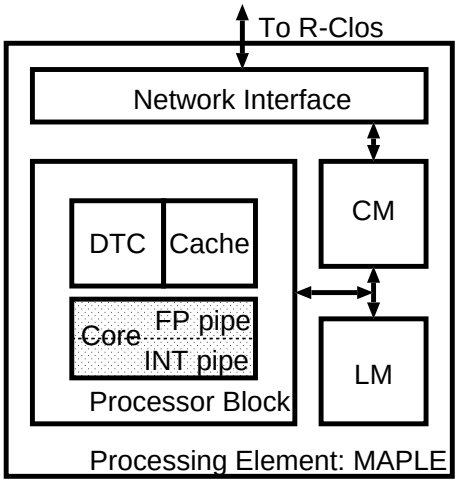

Figure 2: Structure of processing element MAPLE

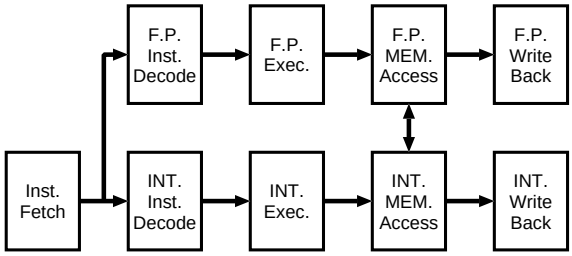

Figure 3: Structure of Pipeline

A processor is further consisting of MAPLE core, Data Transfer Controller(DTC) and cache.

For efficient near-fine-grain parallel processing, MAPLE provides the following facilities:

- predictable operation time for static scheduling,
- receive registers for quick interprocessor communication,
- light weight barrier and non-synchronization mode for eliminating common synchronization codes, and
- software controlled cache managed with the DTC.

3.2 MAPLE Core

MAPLE core is a 32-bit RISC processor which provides a simple structure with highly predictable operations. Its instruction set is an extension of that of DLX[6]. Like DLX, it has 32 registers each for integer and floating point. As shown in Figure 3, it has a 5-stage pipeline structure and every operation can be executed in a fixed clock cycles. The floating-point execution unit is fully pipelined except the divider, and supports 32-bit/64-bit IEEE std 754-1985. Dynamic optimization techniques like dynamic instruction scheduling are excluded so as to enable precise static scheduling. Out of order execution/completion is only allowed when the execution time of instructions are predicted.

3.3 Mechanisms for near-fine-grain processing

3.3.1 Receive registers

MAPLE provides two types of the fast data transfer mechanism.

For a large size data transfer, MAPLE requests the DMA controller which can load the requested data independently from the MAPLE-core operations. The flag on the local DSM is used to indicate whether the requesting data transfer is completed. This data transfer can exploit a high bandwidth, while it takes a considerable time for setting up.

On the other hand, for a small size one word data transfer required in the near-fine-grain processing, MAPLE has special transfer operations and dedicated registers called receive registers to achieve direct register-to-register transferring.

Figure 4 illustrates the data transfer using receive registers.

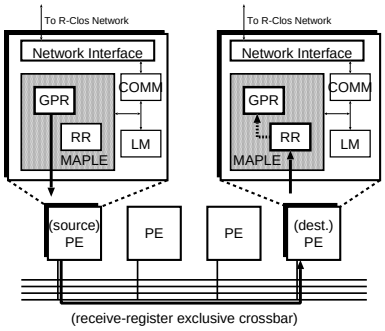

Figure 4: receive register(RR)

16 32-bit receive registers (RR in Figure 4) are provided in the ID-stage of the pipeline, and when the source processor executes a transfer operation, data in a general purpose register in the source processor is directly sent to a receive register of the destination processor through a crossbar. In order to detect the arrival of the required data, there are tag (valid) bits on receive registers each of which is set when the data is received, as shown in Figure 5.

With a receiving instruction, the pipeline is stalled if the valid bit is not set. Otherwise, the data is moved from the receive register to general purpose registers immediately, and the valid bit is reset. For treating receive registers, MAPLE provides two instructions: `sendr`(`sendri`) for sending and `rcvr` (`rcvri`) for receiving.

Although this mechanism can avoid the read-after-write problem, write-after-read problem is not resolved, that is, the data may be overwritten by a new data before reading. In the near-fine-grain parallel processing of ASCA, this problem is solved with the static scheduling or a light weight barrier mechanism.

In the first prototype chip of MAPLE, each receive register supports only one word transferring because of the pin limitation. However, it is sufficient in most data transfer in the near-fine-grain parallel processing.

Compared with a common shared register used in single chip multi-processors Sun Microsystems' MJAC or NEC's MP98[11], receive register is loosely coupled approach and so easy to implement. However, the performance is not degraded, since the synchronization is partly combined.

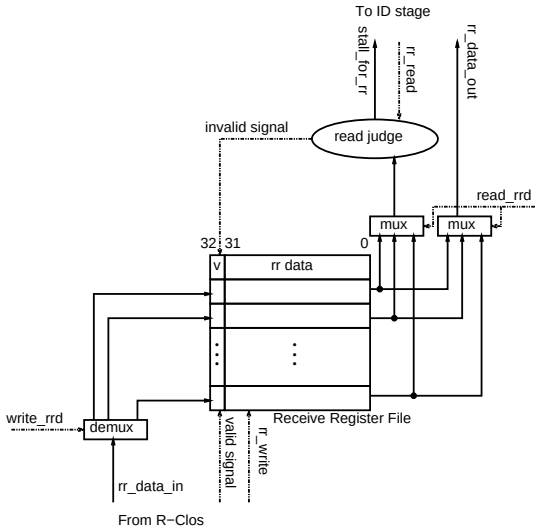

Figure 5: Structure of receive register

3.3.2 Light weight barrier and Asynchronous operation mode

In the near-fine-grain parallel processing, if static scheduling is completely successful, all synchronization codes can be omitted. For this purpose, any optimization techniques which require undeterministic behavior are eliminated in MAPLE core. The memory access is also designed to be deterministic using the software cache supported by the DTC described in the next section. Even though, the network congestion will cause the situation that the prefetched data is not loaded into the cache in time. To cope with this problem, MAPLE has a light weight barrier mechanism, and two operation modes: synchronous/asynchronous.

Each instruction of MAPLE has a few bit synchronization tags, and the light weight barrier synchronization mechanism consisting of a simple open-drain bus is driven. In the synchronous mode, this mechanism is enabled, and processors are stalled until all processors in the cluster executes instructions with the same tag. Using this light weight barrier mechanism. processors can be synchronized without executing instructions dedicated for synchronization.

Two modes in MAPLE is switched as follows using this light weight barrier.

- Usual scheduled codes are executed in the asynchronous mode. In this mode, the light weight barrier is disabled.
- If a processor detects a undeterministic situation (eg. cache miss), it changes its mode into synchronization mode.
- In the synchronization mode, the light weight barrier is enabled, and when all processors are synchronized, the mode returns to the asynchronous mode.

When a cluster of MAPLEs work in this asynchronous mode, it can be treated as a loosely coupled VLIW processor.

3.4 The Data Transfer Controller

The DTC is an intelligent controller which hides the latency for accessing both the shared and local memory. It is also designed suited for the multi-grain parallelizing scheme.

For the coarse-grain parallelism, a large data set transfer of Macro Task (MT) will become a critical overhead. If the transfer of MT data set is completed until the start-up phase of MT, the overhead can be completely hidden. Although it is difficult to be done, the DTC tries as much as possible according to the scheduled code by the compiler. In this case, block transfer using the DMA is requested from the DTC.

On the contrary, since in the near-fine-grain parallelism, frequent communications with a lot of synchronizations between processors will dominate the performance, we adopt a precise static scheduling for the block which does not involve runtime decisions to eliminate these synchronizations. However, in this scheme, an uncertain factor, hit or miss-hit of cache, spoils the precise effective static scheduling. To cope with this problem, a software controlled cache system by the DTC is essential. In the system, data loading and replacement of the cache lines are mainly controlled by the scheduler's generating code so as to realize the always-hit-cache system except for special cases. Also as cache lines are controlled by the scheduler, full-associative scheme is implemented with a small amount of hardware.

The DTC is a simple processor with three-stage-pipeline and has three control modes: software cache, hardware cache + preload/poststore, and hardware cache only. In the software cache mode, the DTC executes instructions generated from the static scheduler. In order to prepare the required data for cache memory before using it, the static scheduler calculates the latency of data transfer and generates the DTC code with main processor's code that invokes the DTC instructions. Since the main processor has a five-stages-pipeline with out of order completion, the precise behavior of the processor is inspected by a pipeline simulator included in the scheduler software. When the static scheduled software cache mode is broken down by some uncertain factors which could not predict in the scheduler, the DTC changes its mode from software cache into hardware cache. After that, the cache behaves as a common hardware controlled cache.

3.5 Operations of the DTC

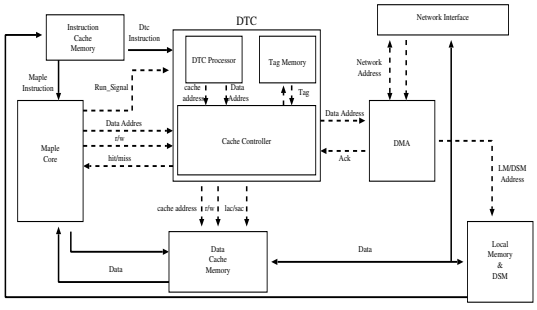

Figure 6: The Structure of Cache

As shown in Figure 6. the DTC consists of a DTC Processor, Cache Controller and Tag memory.

DTC Processor is a simple 64-bit processor with three-stage pipeline and has four instructions to control transferring to/from the cache memory. An instruction is executed by receiving a control signal from MAPLE, and the time when the control signal is issued is buried with MAPLE instructions generated by the compiler. If the DTC instruction is the data transfer operation, it sends a request to the Cache Controller. Although the loading and replacing data are triggered by the DTC instruction, the operation itself is executed in the Cache Controller. Once triggered, the Cache Controller manages the data transfers between memory systems (DSM,CSM and LM).

When the scheduler in the compiler judged that the software cache is not effective, the cache can be also used as a common 4-way hardware controlled cache. In this case, the DTC behaves a simple prefetch controller.

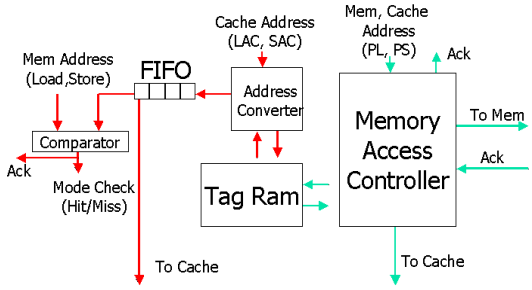

Figure 7: The Structure of Software Cache Controller

Figure 7 shows a part of software cache controller. Right half part of figure shows data flow between the cache memory, the local memory and the external memory. The rest of figure mainly shows data flow between MAPLE core and the cache memory for judging the effectiveness of software cache. In the software cache control mode, all of these data flows are controlled by the DTC instructions generated by static analysis of the compiler.

The DTC Processor has four instructions: Load Address Conversion (LAC), Store Address Conversion (SAC), PreLoad (PL) and PostStore (PS). These instructions work as follows:

- PL prefetches data from a local/shared memory to a cache memory through Memory Access Controller(MAC), and writes the entry in the tag memory at the end of preloading.
- PS transfers data from a cache memory into a local/shared memory through the MAC, and deletes the entry in tag memory at the end of operation.
- LAC converts from a cache address into a memory address through Address Converter and push the both addresses into the FIFO. Since a valid data in corresponding to the cache address stored in the head of the FIFO is on the data bus, the acknowledge to load instruction from MAPLE core requires just one clock cycle when MAPLE core issues a load instruction. The memory address in the head of the FIFO is compared with the memory address of load instruction from MAPLE core and the result is used for judging whether the software cache is broken out.

- SAC operates in the same manner with LAC until queuing. The data from MAPLE core is stored in the cache address specified by the SAC instruction when MAPLE core issues a store instruction.

As long as running on a software cache control mode, this system can realize the most efficient cache utilization, data localization and quick cache access based on the static analysis. Though our goal is that this analysis coincides on real execution perfectly by implementing a processor(MAPLE core) and network switches(R-ClosII) tailored by the static analysis of the compiler, some exceptional dynamic determination still exists. If the comparison of a memory address is false, this cache system behaves as general hardware controlled cache after that.

4 Prototype Implementation

Although four or five processing elements corresponding to a cluster is implemented in a single chip in the near future, a prototype MAPLE is implemented with two prototype chips: MAPLE core and DTC chip.

4.1 MAPLE Core

The MAPLE core chip is implemented on Rhom's 0.35μm CMOS cell-based LSI. Libraries are supported by VDEC Japan. About 80% of gates are used for the floating pipeline, and receive registers and light weight barrier mechanism requires only 6000 gates. Although rather conservative process is used, it works at 80MHz clock.

Figure 8: The Specification of MAPLE Core

Chip	Rohm, CMOS 0.35um poly 2 Metal 3
Maximum clock	80MHz
Gates	174010
The number of pins	466

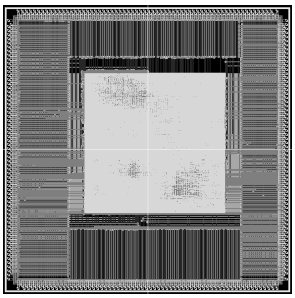

Figure 9: The Layout of MAPLE

Figure 9 shows the layout of the prototype MAPLE chip. Since the required hardware can be reduced, rather small chip area is occupied in real gates.

4.2 DTC

The specification of DTC chip is shown in Table 1. The first DTC chip was implemented on 0.35μ Hitachi Gate Array also supported by VDEC. Since it is a prototype chip with a small amount of gates, a small off-chip cache memory (8K byte) is assumed, and 64 word $\times$ 27 bit $\times$ 4 way tag memory is mounted on the chip. However, the computer simulation results show that the software cache supports better performance compared with hardware cache[10].

Table 1: The Specification of DTC

Chip Feature	Hitachi, 0.35um Gate Array poly 1 Metal 5 190 pins 143k gates
Maximum clock	81.97 MHz
Logics	35,761 BC(7,1522 gates)
Area utilization	50.02 %
The number of pins	185

5 Performance estimation

Performance of single processor The performance of a single MAPLE core is estimated with benchmark programs (FFT and FLOPS) using the clock level logic simulator. FLOPS includes eight subprograms. Each subprogram calculates numerical integration or Maclaurin series expansion in double format. Figure 10 shows the reciprocal of FFT execution time and MFLOPS value at the subprogram which includes 3.4% DIV instruction.

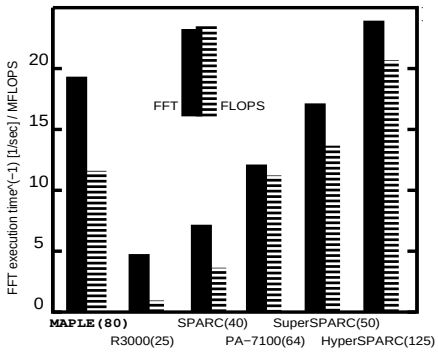

Figure 10: Evaluation of FFT and FLOPS

Processing capacity of MAPLE is comparable to the early super-scalar processors even though it was used as a simple single processor system.

Performance of a cluster (4 MAPLES) Here, we analyzed the performance of a cluster with four MAPLEs when an application program called "Picalc" is executed in the near-fine-grain parallel processing. Considering the hardware requirement, a cluster can be integrated on a chip and comparable to recent high performance microprocessors. Notice that every communication between PEs uses direct register-register data transfer mechanism of MAPLE.

Picalc is a series calculation program to find the value of π with many loop iterations.

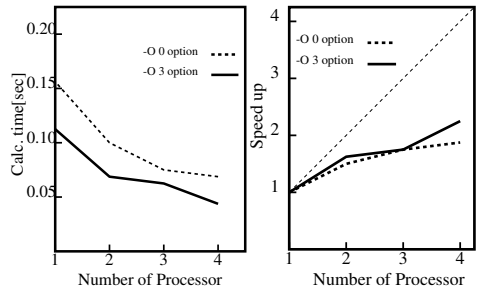

Figure 11: Speedup of execution Picalc

Figure 11 shows speedup against the number of PEs. With compiler's optimization options, useless codes are removed so that large speedup rate is obtained and the performance of 4 PEs is 2.25 times higher than that of single PE.

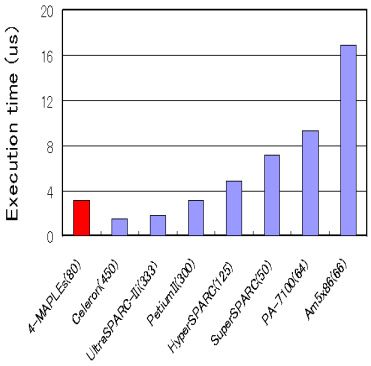

Figure 12: Performance of a cluster

Figure 12 demonstrates that the performance is comparable to recent high performance processors when a cluster is pushed into a single chip. Note that the power consumption is much reduced, since the clock frequency is much less than those of recent microprocessors.

6 Conclusion

The processor architecture dedicated for efficient execution of near-fine-grain parallel processing is proposed, implemented and evaluated. Performance evaluation based on a real design shows that the performance of a cluster consisting of MAPLE processors is comparable with recent high end super-scalar processors in spite of its simple structure and low frequency operation.

Now, two prototype chips described in this paper: MAPLE core and the DTC are available. The print circuit board which mounts these chips, memory, and interfaces are now under developing. Using the boards, a multi-processor corresponding to a single cluster can be built. Simulation studies of multi-cluster systems including static scheduler are also our future work.

References

[1] Okamoto,M., Yamasita,K., Kasahara,H. and Narita,S., "Hierarchical Macro-Dataflow Computation Scheme on a Multiprocessor System OSCAR", Proc. IEEE Pacific Rim Conference on Communications, Computers, and Signal Processing, pp.44-49, May. 1995. Scheme of Fortran Programs

[2] Constantine Polychronopoulos, Milind B. Girkar, Mohammad R. Haghighat, Chia L. Lee, Bruce P. Leung, Dale A. Schouten "Parafrase-2: An Environment for Parallelizing, Partitioning, Synchronizing, and Scheduling Programs on Multiprocessor", Proceedings of the International Conference on Parallel Processing, St. Charles IL, August 1989, pp. II39-48

[3] Ogata,W., Fujimoto,K., Oota,M. and Kasahara,H., "Compilation Scheme for Near Fine Grain Parallel Processing on a Multiprocessor System without Explicit Synchronization", Proc. IEEE Pacific Rim Conference on Communications, Computers, and Signal Processing, pp.327-332, May. 1995

[4] K.Iwai, T.Morimura, T.Fujiwara, K.Sakamoto and H.Amano, "An interconnection network of ASCA: a multiprocessor for multi-grain parallel processing", Proc. of 6th IASTED Symposium on Applied Informatics, pp.255-257, Feb.1998

[5] T.Fujiwara, T.Kawaguchi, K.Sakamoto, K.Iwai, H.Amano, "Custom Processor for the Multiprocessor ASCA", Proc. of 6th IASTED Symposium on Applied Informatics, Feb. 1998

[6] John L. Hennessy and David A. Patterson, "COMPUTER ARCHITECTURE A QUANTITATIVE APPROACH SECOND EDITION", Morgan Kaufmann Publishers, 1996.

[7] Tomohiro Morimura, Keisuke Iwai, Hideharu Amano, "Multistage Interconnection Network Recursive-Clos(R-Clos) : Emulating the hierarchical multi-bus", PDCS '98, pp.99-104, Sep.1998

[8] Tomohiro Morimura, Kensuke Tanaka, Keisuke Iwai, Hideharu Amano, ""Multistage Interconnection Network Recursive-Clos(R-Clos II) : a scalable hierarchical network for a compiler directed multiprocessor ASCA", PDPTA 2001

[9] T.Abe, T.Morimura, T.Suzuki, K.Tanaka, M.Koibuchi, K.Iwai, H.Amano, "ASCA chip set: Key components of multiprocessor architecture for multi-grain parallel processing", Proc. of COOL Chips IV, pp.223-247, Apr.2001

[10] K.Iwai, T.Morimura, T.Kawaguti, A.Sakai, T.Abe, H.Amano, "ASCA: A multiprocessor architecture", JSPP2000, pp.2-10, June.200

[11] http://www.labs.nec.co.jp/MP98/, "MP98 Project web page"