A performance evaluation of the multiprocessor testbed ATTEMPT-0

Takuya Terasawa\textsuperscript{a}, Ou Yamamoto\textsuperscript{b}, Tomohiro Kudoh\textsuperscript{c} and Hideharu Amano\textsuperscript{b}

\textsuperscript{a} Dept. of Information Networks, Tokyo Engineering University, 1404-1 Katakura-cho Hachioji 192 Japan
\textsuperscript{b} Dept. of Computer Science, Keio University, 3-14-1 Hiyoshi Kohoku-ku Yokohama 223 Japan
\textsuperscript{c} Dept. of Information Technology, Tokyo Engineering University, 1404-1 Katakura-cho Hachioji 192 Japan

Correspondence to: T. Terasawa,
Dept. of Information Networks, Tokyo Engineering University, 1404-1 Katakura-cho Hachioji 192 Japan
Phone: +81-426-37-2111 (ex.2610), Fax: +81-426-37-2118, E-mail: terasawa@cc.teu.ac.jp

Abstract

ATTEMPT-0 is a bus connected multiprocessor testbed which provides both the shared memory and the local memory for each processor. In this paper, we describe the design policy and organization of machine and discuss on the performance measurement results using five practical parallel applications. Processors share a global shared memory through the Futurebus. A write-through type snoop cache, a local memory and a special synchronization mechanism synchronizer are provided for each processor. The evaluation result demonstrates the efficiency of the cooperated system with the local memory, the synchronizer and the global shared memory. By using the local memory, performance is improved 10 to 60\% compared with the case of only the shared memory is used. Meanwhile, through the performance analysis of the Futurebus, we found some problems in the arbitration protocol.

Keywords. Bus connected multiprocessor; snoop cache; synchronization mechanism; IEEE Futurebus; performance evaluation.

1 Introduction

A bus connected multiprocessor is one of the most promising types of small scale parallel machines because of its simple and economical structure. A small number of processors are connected with a shared memory through a bus. Usually, all processors share a common address space of the shared memory.

Since this structure is a straight extension of a uniprocessor, the operating system can be implemented with a little modification of the traditional operating system like UNIX. Processes of the traditional operating system are simply distributed to idle processors when they are invoked. However, in this structure, the congestion of the shared bus or shared memory dominates the system performance. In order to cope with this problem, each processor provides a private cache with a snoop mechanism\cite{1}. Since the control algorithm for maintaining the coherence of the snoop cache is a key of the performance, many kinds of coherence protocols\cite{2}\cite{3} are proposed. Most of recent commercial machines use simple invalidation type protocols like Illinois\cite{4}, and most of recent microprocessor chips provide a controller for the snoop cache inside the chip.

Unlike uniprocessors, inter-processor communication and synchronization are essential in multiprocessors. However, in most multiprocessors, the snoop cache which is, in a sense, an extension of a uniprocessor cache manages everything including instructions, stacks, shared data for inter-process communication, and variables for synchronization. Since their ways to be accessed are much different with each other, the communication between caches (or memory and cache) often degrades performance when data exchange and synchronization are frequently requested. Although recent multiprocessor workstations are commercially successful, most of them are not used for the parallel execution of a single job but for the fast execution of independent jobs.
In order to execute a single job efficiently in parallel on these machines, cooperation of the local memory, cache, and communication/synchronization mechanisms are desirable. We started a project called ATTEMPT (A Typical Testing Environment for MultiProcessor sysTem) to develop a multiprocessor with such a combined universal communication mechanism in 1988 when the first generation of bus connected multiprocessors (e.g. Balance-8000 or Multimax[5]) were becoming commercially successful.

When this project started, we classified inter-processor communication into two categories: the active communication including real data exchange with/without synchronization and the passive communication represented by the object codes and local variables[6]. In order to support the active communication, a high performance data exchange mechanism called the synchronizer is proposed. For the passive communication, the cooperation of the local memory and the private cache is effective. For a message passing used in the object oriented languages, Keio protocol which allows the use of a cache line as a message buffer is also proposed[6].

For evaluating this concept on the real machine, a prototype multiprocessor called ATTEMPT-0 was developed in 1989 (Figure 1). A lot of parallel applications including a logic simulator[7], LSI channel router[8], neural network simulators[9], and circuit simulator[10] were developed on this prototype.

Approaching to the goal of the project, ATTEMPT-0 provides a combined mechanism of local memory, snoop cache, and the synchronizer. The shared bus of ATTEMPT-0 is the IEEE Futurebus[11] which was accepted as the IEEE standard in 1987.

However, the snoop cache protocol actually implemented on ATTEMPT-0 was a simple write-through instead of our Keio protocol, and some design mistakes which lost a balance of the system were found. Now, a next prototype ATTEMPT-1 is under development based on the experience of ATTEMPT-0. Considering the implementation of a multiprocessor on a single VLSI chip, ATTEMPT-1 provides a multi-protocol cache which also provides functions of the synchronizer used in ATTEMPT-0.

Now, ATTEMPT-0 is almost finishing its role, and it is not a complete successful machine. However, through the experiences with this machine, the followings were quantitatively measured.

- How much performance is improved with a combination of the synchronizer, cache, and local memory.
- Since the IEEE Futurebus was quickly modified to Futurebus+[12], ATTEMPT-0 is one of a few multiprocessors with the old Futurebus. The performance of this bus and arbiter which is one of the most interesting mechanism in this bus protocol were evaluated.

In [13], Hennessy and Patterson say: There is value in projects that do not affect the computer industry because of lessons that they document for future efforts. The sin is not in having a novel architecture that is not a commercial success; the sin is not quantitatively evaluating the strengths and weaknesses of the novel ideas.

Here, we report our experiences on ATTEMPT-0 for future efforts. In section 2, the goal and concept of the ATTEMPT project are introduced. In section 3, the structure of ATTEMPT-0 is described. The application and their implementation are introduced in section 4. And in section 5, we discuss the result of the evaluations. In section 6, the performance of the Futurebus is described. Finally, we give a conclusion in section 7.

2 ATTEMPT project

2.1 The goal of the project

Even in a uniprocessor, program codes, stacks, and global data such as an array are accessed in different ways. In multiprocessors, the use of shared variables for synchronization and data exchange much increases the variety of access patterns. For example:

- Program codes and local data in the stack are accessed continuously by a specific processor on which the corresponding process is running.
- Small sized variables for synchronization are frequently accessed by various processors.
- Large global shared data such as an array for matrix calculations is accessed by various processors with less frequent synchronizations.
Message passing used in the object oriented languages require one to one communications between processors. The size of a message is usually large.

Ever since the project started in 1988, most of bus connected multiprocessors support all types of above accesses only by shared variables on a snoop cache with invalidation type write-through or write-back protocol. Since the cache with invalidation type protocol is suitable for sharing the program code and local data, processes which communicate less frequently can be efficiently executed. However, parallel programs which require frequent data exchanges yield poor performance unless the number of processors is small.

In ATTEMPT project, we proposed the synchronizer which supports multicast of a single word data and synchronization with interrupt. Using the synchronizer, small data exchange with synchronization is efficiently supported. In addition, a novel cache protocol called Keio protocol is proposed to support a large sized message passing. Furthermore, we prepared a local memory for each processor. If program code and local data can be allocated on the local memory of the processors, the bus congestion can be further reduced.

Using these facilities, data can be allocated as follows:

- synchronization variables, small sized messages, small sized shared data which requires frequent communication: synchronizer,
- program code, stack, local data: local memory,
- large sized messages, processes which require frequent migrations, and other data: snoo cache with Keio protocol.

The goal of ATTEMPT project is to build a relatively large sized bus connected multiprocessor (about 20 processors) with the above facilities, and to execute parallel programs with processes which requires frequent data exchanges without performance degradation.

### 2.2 Synchronization mechanism

The synchronizer is a multicast memory with inter-processor interrupt mechanism. It supports both multicasting of data and synchronization with/without interrupt efficiently with the distributed implementation.

#### 2.2.1 Data multicast using the synchronizer

A synchronizer consists of synchronizer units each of which is attached to a processor. As shown in Figure 2, a synchronizer unit provides a synchronizer memory associated with the following flags:

- registration flag (R),
- write interrupt flag (WI) for write operation, and
- zero interrupt flag (ZI) for Fetch&Dec operation.

These flags are set or reset only by the processor to which the synchronizer unit is attached.

When a processor writes a data into its synchronizer memory, the address and data are put on the shared bus. All synchronizer units receive the accessed address from the shared bus, and refer the registration flag. If the flag associated to the written address is set, the data is stored into the synchronizer memory. If this flag is reset, all operations for the address of the synchronizer are ignored. Thus, the data is multicast to only synchronizer units which associated registration flag is set. The read operation from the synchronization memory is executed without using the bus. This mechanism works like a cache with an update type protocol from the viewpoint of data multicasting.

#### 2.2.2 Synchronization operation

The Fetch&Dec(X) operation, which is a synchronization operation in the class of Fetch&Φ proposed in [14], can be executed for every variable stored into the synchronizer. It is also implemented in the distributed manner. If X is not zero, the value of X is fetched and indivisibly decremented. As shown in Figure 3, each synchronizer unit locks the bus first. Then, it returns X to the processor, and at the same time, X-1 is sent to the bus through the decrementing hardware.
If X has been zero, it is never decremented no longer, and only the value zero is obtained by this operation. In this case, the bus is not utilized. Therefore, processors in the busy-waiting state do not influence the bus.

2.2.3 Inter-processor interrupt mechanism

For sending interrupt requests between processors, a pair of flags – a write interrupt flag and a zero interrupt flag – is used. When a processor executes the write operation for variable X in the synchronizer, the interrupt request is sent to all processors whose associated write interrupt flag is set. The zero interrupt flag is used for interrupt request caused by the Fetch&Dec(X) operation. If X becomes zero as the result of the Fetch&Dec(X) operation, interrupt signals are sent to processors whose associated zero interrupt flag has been set.

These primitive operations support synchronization among processors. In Figure 4, some examples of the synchronization are shown.

Ex.1. Mutual Exclusion

In this example, mutual exclusion is realized. The value of the X is always 0 or 1.

Ex.2. Synchronization of message multicast

This example uses interrupt to notify the processor of the status. The sender sets (enables) the zero interrupt flag in advance, and the receiver sets the write interrupt flag in advance. The receiver is notified by the interrupt that the message is ready. When the nth receiver process receives the message, the synchronizer X becomes zero by the Fetch&Dec(X) operation. At that time, interrupt is transferred to the sender process in order to inform it that the next message can be sent.

2.3 Cache control mechanism

Our goal is to achieve the maximum performance with a combination of the synchronization mechanism, snoop cache, and the local memory. The synchronization variables, small size messages, and data which is frequently updated are supported by the synchronizer, while instructions and local data are located on the local memory. Thus, the cache is designed to support other types of accesses: large size messages, large arrays of data, and other data which cannot be supported by the synchronizer nor local memory.

Many cache coherence protocols have been proposed and evaluated [2][3][15]. We adopted a write invalidation type protocol, and proposed a novel cache control protocol called Keio protocol which enables effective use of a cache line as a message buffer [6]. However, this protocol is not implemented in ATTEMPT-0 which is focused in this paper because of its complicated control mechanism.

In ATTEMPT-0, the simple snoop cache with the write-through protocol is used. Although a number of researches demonstrated that the performance of the write-through protocol is much inferior to that of the write-back protocols, we can use the synchronizer and the local memory for data which cannot be supported efficiently by the write-through cache. Our evaluation results demonstrate how synchronizer and local memory can reduce the overhead of the simple write-through cache.

3 Organization of ATTEMPT-0

Figure 5 shows a block diagram of ATTEMPT-0. The backplane bus allows to connect maximum of 20 boards.

**Processor** Motorola MC68030 and MC68882 are used as the CPU and the FPU, respectively. Both of them are driven by a common 20MHz clock on a single board. A 64Kbytes ROM holds a simple monitor program, and a 128Kbytes static RAM (SRAM) is provided for the stack, instructions and other local data. In addition, a 4Mbytes dynamic RAM (DRAM) is provided as a local memory. The local memory is mainly used as an area of instructions, local data and heaps. By swapping jumpers on the board, the local memory can be used as a shared memory. In our measurement, two boards are used for 8Mbyte shared memory in total.

**Cache** As a cache controller, we used NEC’s µPD71641. This controller has a snoop mechanism with simple write-through protocol. In this protocol, when a processor writes into a shared memory location, all corresponding cache lines of other processors are invalidated, and the data is always written into the shared memory directly. The controller has a dual-port tag memory for the simultaneous accesses from the processor side and the bus side. However, the cache memory itself is not a dual port one. An interface between the controller and the bus
is controlled by three PLD chips. Both of the interface and the cache controller are driven by the 20MHz clock, which also drives the CPU and the FPU. The cache is 2-way set associative, and replaces lines with the LRU policy. The size of the cache memory is 64Kbytes and the cache line size is 64 bytes.

**Synchronizer** A synchronizer described in the previous section is managed by two independent controllers each of which consists of two PLD chips. One of the controllers manages sending request from the attached processor, while the other manages receiving the data from the bus. The size of the synchronizer memory (without flags) is 32Kbytes.

**Backplane bus** We use the IEEE Futurebus (IEEE P896.1 1987)[11] as a backplane bus. Futurebus supports fully distributed and asynchronous protocols on the data transfer and the bus arbitration. Especially, a protocol for the bus arbitration is performed completely in a distributed manner. The data transfer and the arbitration can be processed in parallel. Both of address and data have 32bit width and are multiplexed on the bus. On ATTEMPT-0, the performance of the data transfer is 26Mbit/sec.

**I/O interfaces and other parts** Each processor board has a RS-232C interface and a GPIB interface. SCSI and Ethernet interfaces and a timer also can be connected through a board-local VME bus. In our current configuration, ATTEMPT-0 is connected to the host workstations through the GPIB and the Ethernet interfaces. A hard disk drive, which is used for the memory swapping, is attached through the SCSI interface.

**Hardware monitor** A hardware monitor board, which is attached to a processor board through the board-local VME bus, measures various events in the system without affecting any operation of processors. The board is used for the performance evaluation of the machine and tuning of software.

**Software** A monitor program on the ROM supports the communication with the host machine and the handling of the interrupts. It also has the facilities for loading application programs and debugging them. The application programmers can utilize a user-level library for operations of the I/O and synchronizer. Various software systems including C-Threads of Mach operating system, the memory swap system between the shared memory and the hard disk, and the virtual memory system for the local memory have been implemented on ATTEMPT-0. However, in order to eliminate the loss in the operating system, they are not used in the evaluation described here. Programs are directly written with user-level library calls, and executed with the simple monitor system supported on the ROM.

Table 1 shows the basic parameters including access times of various memory systems of ATTEMPT-0. Both the access to the local memory and the transfer of the cache lines have a large latency. In ATTEMPT-0, a DRAM on each processor board can be utilized both as a shared memory and a local memory by swapping jumpers on the board. With this design strategy, the DRAM system cannot be optimized for the shared memory nor local memory. This causes a large latency of DRAM memory system especially for the block transfer used in the replacement of the cache lines.

4 Applications

4.1 Application programs

We selected five practical parallel applications for our evaluation. Table 2 shows the outline of each application. Two of them, Channel[8] and Logique[7], are our original programs developed on ATTEMPT-0. Channel solves the channel routing problem of the VLSI layout based on Hopfield type neural network. Logique is a parallel logic simulator based on a Chandy-Misra’s distributed simulation algorithm. Since these programs are written for ATTEMPT-0, the data distribution is well considered and fully optimized.

Other three, Cholesky, MP3D, and Water, come from SPLASH[16] benchmark programs developed in Stanford University for multiprocessor benchmark. These programs are for multiprocessors without local memory.

In most of execution results described in later, only shared data is allocated on the shared memory. Local data and stack are allocated on the local memory attached to each processor. ATTEMPT-0 provides two local memory modules: small but high speed SRAM and large but low speed DRAM. Except Channel which requires a large stack area, program code, static data and stack are allocated on the SRAM and the DRAM is used for dynamically allocated variables. On-chip instruction cache is enabled in all executions.
4.2 Parallel execution of application programs

Before the discussion on architectural trade-off, the execution results and behavior of parallel programs are briefly presented here.

Figure 6 shows the speedup ratio compared with the execution time with a single processor. It shows that the execution time of programs except MP3D is reduced as the number of processors increases. Figure 7 shows the cache hit ratio. Although the hit ratio of some applications decreases when the number of processors is large, high cache hit ratio (more than 95%) are maintained in most cases. Figure 8 and 9 show the bus utilization ratio and the ratio of waiting time for bus mastership. These figures indicate the degree of congestion on the system bus and thus, on the shared memory. They show that the bus and the shared memory bottleneck the system performance in some applications while the congestion of them is not severe in others.

From these figures, the behavior and characteristics of application programs are analyzed.

Channel Since this program is optimized for ATTEMPT-0, it shows the lowest bus utilization ratio and the lowest waiting time ratio (Figure 8 and 9). In this program, only output values of neurons which are required by other processors are located on the shared memory. A processor reads values from the shared memory, calculates using data on the local memory, and then update the shared values. Although invalidation loss causes the degradation of cache hit ratio with a large number of processors (Figure 7), it does not increases the bus utilization so much. Nevertheless, the speed up ratio is worse than those of Logique and Water (Figure 6) because the imbalance of the load is not solved in this program[8].

Logique As shown in Figure 6, the speed up ratio of this program is “super-linear”. It comes from the discrete time management system of this program. When the number of processors increases, the length of the scheduling list for each processor is reduced. The time for searching the scheduling list is drastically reduced resulting the “super-linear” speedup. Since the program requires frequent access to the scheduling list on the shared memory, the bus utilization becomes large with a large number of processors.

Water Although this program is not written for a multiprocessor with local memory systems, the bus utilization and bus waiting time are close to those of Channel that is written for ATTEMPT-0 (Figure 8 and 9). It comes from the large granularity of this application. That is, the access to the shared memory is not so frequent compared with the time of calculation. Moreover, since the calculation load is well balanced in this program, the speed up ratio is better than that of Channel (Figure 6).

Cholesky The speedup ratio of this program is not so good (5.8 times with 10 processors, Figure 6) compared with others. From the analysis of the calculation and communication ratio (Figure 13), it appears that the time for write operations to the shared memory occupies a large fraction in the total execution time. Since the cache hit ratio of this program is kept high as shown in Figure 7, the bus congestion caused by the frequent write operations prevents the performance improvement.

MP3D This is the hardest one to execute among the five applications used here. In MP3D, the cache hit ratio is the worst as shown in Figure 7, because of frequent updating of the shared data. Since the low hit ratio introduces a large amount of replaces, the bus waiting time overcomes the calculation time when the number of processors increases (Figure 14). This figure shows that the write accesses to the shared memory is also large. As a result, the performance is not improved more than twice.

5 Discussions on the evaluation results

5.1 Effect of the local memory

In order to evaluate the performance improvement with the local memory, we evaluated the execution time and bus congestion with two types of data assignment. Here, ‘all-share’ refers to the case when all the data including program code, private data and stacks are placed on the shared memory, and ‘use-local’ refers to the case when both the shared memory and the local memory are used. In order to avoid the confusion, we represent the results of, for example, MP3D with ‘all-share’ like MP3D(AS) while those with ‘use-local’ like MP3D(UL).
Since ATTEMPT-0 equips write-through type caches, accesses to stacks cause heavy contention on the bus. Therefore, for Cholesky, Cholesky(ULS) in which only stacks are placed on the local memory is also evaluated.

Figure 15 shows the execution time of each application in 'use-local' and 'all-share'. The performance of Cholesky(UL) is 60% better than that of Cholesky(AS), and 20% better than that of Cholesky(ULS) with 10 processors. The difference of the performance increases along with the increase of the number of processors (Figure 16). MP3D shows less significant differences, while MP3D(UL) is 16% faster.

Analysis results of the execution time are shown in Figure 17 through 19. Compared with the 'use-local' versions, the ratio of the bus waiting time is increased in 'all-share' versions. Figure 20 also shows the heavy bus congestion of 'all-share' versions. Note that the ratio of the time for write operations to the shared memory is especially large in 'all-share' as shown in Figure 21.

These results suggest that the main reason of the performance improvement when local memory is used is the elimination of the bus congestion caused by stack accesses. Since the write operations always generate bus accesses in write-through policy, placing stack on local memory greatly reduces bus congestions.

5.2 Effects of synchronizer

5.2.1 The impact of differences in the implementation of barrier synchronization

To study the effect of the special synchronization mechanism, synchronizer, barrier synchronization is implemented in three different manners for MP3D, Cholesky and Water.

(1) The Fetch&Dec function of synchronizer is used. Only one processor writes the number of remaining processors to a synchronization variable, and the other processors execute the Fetch&Dec operation to this variable, and then wait for the variable to be zero. This method is standard in ATTEMPT-0 and used in evaluations in other sections.

(2) A variable for counting the number of processor is placed on the shared memory. Each processor once decrements the variable and waits for it becoming zero by other processors' activities. Fetch&Dec is used for the mutual exclusion of accesses to the variable.

(3) A bit-vector in which each bit of the data corresponds to each processor is placed on the shared memory and used for the barrier synchronization. Actually, a byte-vector is used since a byte is the smallest access unit. synchronizer is not used.

As shown in Figure 22, the performance of MP3D is improved by using the synchronizer when more than 4 processors are used. The counter variables or byte-vectors on the shared memory are repeatedly shared and invalidated. This causes many replacements of the cache lines. As we mentioned in 5.3, differences among the methods are appeared since MP3D is severely influenced by the replacements of the cache lines. In other two applications (Water and Cholesky), since the bus congestion is not so severe compared with MP3D, no performance improvement is observed.

5.2.2 Allocation of shared data to the synchronizer

In this section, we discuss on the performance improvement by allocating frequently accessed shared variables on the synchronizer. The synchronizer is used as a kind of multicast memory. We applied this method for Logique and MP3D which include frequently updated shared variables. (Cholesky also has such variables, but they are too large for the synchronizer.) The target circuit of Logique is 8bit ALU (124 gates, 560 events), and the shared data structure called message queue is placed on the synchronizer. In MP3D, the target problem and conditions are same as which used in the previous sections, and data for reservoir space and boundary conditions are placed on the synchronizer.

The comparison of the execution time between original programs which use the shared memory and the modified versions which use the synchronizer is shown in Figure 23.

In Logique, the performance is saturated in 5 processors due to the shortage of available parallelism of the small target circuit, but the performance is improved 5 to 10% when the synchronizer is used. In MP3D, the improvement is 5 to 10%, and the difference of the execution times increases according with the number of processors increases.

The reason of the poor performance improvement is the insufficient synchronizer memory space. In ATTEMPT-0, the synchronizer is only 32Kbyte, and only a part of the shared variables can be allocated. If larger data structures such
as space cells in MP3D which is the main cause of the performance degradation, can be allocated on the synchronizer, a larger performance improvement can be obtained.

5.3 Effect of the cache control policy

Performance evaluations of snoop cache systems have shown that the performance of the write-through policy is much less than those of the write-back policy. Archibald and Baer’s evaluations in probabilistic models[2] shows that the performance of a multiprocessor with the write though cache is never improved when the number of processors exceeds 5 or 6. Annaratone and Rühl measured the write-through policy and the write-back policy on the real machine (Sequent’s Symmetry)[17], and concluded that the performance of the write-back is 20% to 50% better than that of the write-through cache.

Although the direct comparison between the write-through and write-back cache is impossible on ATTEMPT-0, our results show that the performance of the write-through is not so bad when it is used with the local memory. As shown in Figure 6, except MP3D, performance is improved even with 10 processors.

Figure 24 shows the hit ratio of ‘use-local’ and ‘all-share’ for MP3D and Cholesky. In MP3D(UL), the cache hit ratio is lower than that of others. The amount of the read misses is larger than that of write misses. Compared with ‘use-local’, the hit ratios of ‘all-share’ are high in both applications. Since in ‘all-share’, the program code and the private data such as stack are placed on the shared memory, these data are also stored into the cache. So, the amount of the actual shared data in the cache memory is decreased relatively, and the effect of the invalidation is reduced. Nevertheless, even in ‘use-local’, the hit ratio of the write though cache is kept larger than 99% except MP3D.

The reason of the large miss ratio of MP3D is as follows. Since the space cells are not assigned to the processors statically, the access locality is small. When the particles are moved to the other cells, the cell data is shared among processors resulting a large number of cache misses due to the invalidations. These cache misses cause many replacements of the cache lines, therefore, the high bus utilization ratio and the long bus mastership waiting time are introduced. However, this type of cache miss cannot be saved even with the common write-back cache with the invalidation policy. These results suggest that the update policy is suitable for the access pattern like MP3D.

6 Performance analysis of Futurebus

Futurebus is a standard backplane bus for high performance multiprocessor systems and was approved as an IEEE standard in 1987 [11]. At that time, Futurebus provided following attractive features for our machine, ATTEMPT-0:

1. architecture independence accomplished by providing users with relatively abstracted specifications,
2. a solution to the problems on bus driving such as settling of bus signals,
3. a completely asynchronous protocol which uses both edges of handshake signals, and
4. useful facilities for locking and mutual exclusion in multiprocessing environment.

In this section, we evaluate two important factors for the bus system, the performance of data transfer and bus arbitration.

6.1 Performance of Data Transfer

National Semiconductors’ BTL (Bus Transfer Logic) DS3897 dedicated for Futurebus transceiver/receiver is used in ATTEMPT-0. It changes the voltage level between the TTL level (5V) and bus level (2.1V) which allows both a small impedance termination and a small rising up delay. Unlike the successor devices for Futurebus+, it requires the CR filters to eliminate the wired-or glitch on the bus when it is utilized for handshake lines. Table 3 shows delay times measured on a bus line of ATTEMPT-0.

The delay time between an input of the transceiver device and an output of the receiver device (“device and line” in Table 3) is about 26ns. A propagation delay of the device itself is typically 10ns, and so the delay inherent in the bus line is about 6ns. This value is smaller than the maximum value (12.5ns) specified in the standard.
Since we use a conservative CR filter to reduce the glitch, the maximum propagation delay between an input of the glitch filter and an output of the receiver device ("filter" in Table 3) is large (82ns). The delay inherent in the glitch filter is $82\text{ns} - 26\text{ns} = 56\text{ns}$. This value is larger than the maximum value (45ns) specified in the standard.

As a result, the performance of the data transfer extracted from above values is about 26Mbit/sec.

6.1.1 Bus Arbitration

When a module connected to the bus wants to send data to, or to obtain data from another module, it must first get mastership of the bus. However, there is a case that two or more modules request the mastership at the same time. Bus arbitration is a scheme to resolve such a contention and to restrict bus mastership to exactly one module at a time. The protocol of the bus arbitration in Futurebus is a fully distributed and asynchronous one. The sequence of the arbitration is controlled by a bus arbiter placed on each module which has an ability to get the mastership. This protocol is very precise, however, as mentioned later, we found that it may introduce an overhead in some cases.

A module which has mastership of the bus is called a bus master. And a module which has the ability to become the bus master is called a potential master. Each potential master has a bus arbiter and a unique number called an arbitration number. A potential master starts the bus arbitration on detecting an internal request for the bus mastership. When the bus arbitration starts, competing potential masters apply their arbitration numbers to the bus, then a potential master which has the highest arbitration number among the competitors becomes the bus master. The bus arbiter consists of two parts. One part is a combinational circuit which decides the bus master using the arbitration number. The other part is a controller for the sequence of the bus arbitration.

6.1.2 Arbitration protocol in Futurebus

Each arbitration number includes a unique number assigned to each connector for a module on the backplane. This number is called a geographical address. If the module placed on higher geographical address always wins the competition, weaker modules will never get the bus, and be in the "starved".

In order to avoid the starvation, Futurebus provides the facility called a fairness scheme. When the bus master relinquishes the mastership, it is prohibited from participating in the arbitration until there are no modules requesting the mastership. The state in which modules are prohibited from participating in the arbitration is called an inhibited state. The arbitration protocol provides a sequence for escaping from the inhibited state. This sequence is called a fairness release. The fairness release occurs only at the time when there are no modules requesting the mastership except modules in inhibited state.

However, this mechanism has two problems: first, the fairness is not completely accomplished. The module with higher geographical address will have more chance to use the bus than those with lower address. It will cause the imbalance of the bus waiting time resulting the performance degradation. Second, as shown in Table 4, a period of the bus arbitration including the fairness release is about 30% longer than that of the arbitration itself. Thus, fairness release cycle itself may introduce the overhead to the arbitration cycle.

6.1.3 The influences of the weak fairness

First, we measured waiting time for acquiring the mastership of a module which has the highest arbitration number (the strongest module) and the lowest one (the weakest module). Here, the waiting time is defined as a period between the issue of a request for the bus mastership and the acquisition of it. Figure 25 shows the result.

In the case of 8 processors, the waiting time of the weakest module is 16% to 33% greater than that of the strongest module. It stretches the calculation time of the weakest module, and if all processors must wait for the completion of the weakest module, total computation time might be stretched.

6.1.4 The overhead of the fairness release cycle

Next, we examined the overhead introduced by the fairness release. The fairness release and the arbitration can be overlapped with bus master’s transactions. However, in the case that the period of master’s transaction is shorter than the period of the arbitration or in the case that the arbitration occurs after the completion of master’s transaction, the overhead is introduced. However, it is difficult to measure this overhead directly even with our monitoring hardware since the clock level counting is required to measure the overlap of the arbitration cycle and the data transaction.
Instead of the direct measurement, we measured the total number of the occurrence of the fairness release and arbitration, and calculated the ratio of the fairness release against the arbitration and fairness release itself. That is:
\[
\frac{\text{# of fairness release}}{\text{# of fairness release} + \text{# of arbitration}}
\]

This ratio, called the fairness release ratio, shows the frequency of the fairness release cycle, and thus, indicates the overhead. If there are two processors which uses the bus in the equal probability, a fairness release is required when the mastership is moved. Therefore, the fairness release ratio becomes 50%. Although it depends on applications, this ratio is reduced if the number of processors increases. If the probability of using bus is equal among processors, the fairness release ratio becomes \(1/N \times 100\%\), where \(N\) is a number of processor. If the probability of using bus is not equally distributed, the ratio becomes large, but never exceeds 50%.

Figure 26 shows the fairness release ratio. In all applications in our measurement, every synchronization requires all processors to participate in it. Therefore, concerning only the synchronization, the fairness release ratio is \(1/N \times 100\%\). However, since the accesses for synchronization occupy a little fraction of whole accesses to shared data, the fairness release ratio is mainly affected by the imbalance of the frequency of the requests for the bus master. For example, in MP3D, all processors have the same shared data. On the other hand, in Water or Cholesky, different processor groups have different shared data. Therefore, Water and Cholesky have higher fairness release ratio than that of MP3D. From Figure 26, the ratio of MP3D is close to \(1/N \times 100\%\), and Water and Cholesky remain near the 50%.

6.1.5 Parking

In Futurebus, a bus master which have completed its bus transaction can remain in the bus master state until the other modules request the mastership. This case is called the "parking". A bus master which is parking can use the bus without any arbitration.

In order to examine the efficiency of the parking, we measured the ratio of the parking in the whole bus requests including the parking itself. We call this ratio the parking ratio. Figure 27 shows the parking ratio of each application.

In Figure 27, the parking ratio goes down as the number of processor increases. However, on every application, the ratio is large with 2 or 3 processors. Especially, in Water, the parking ratio is large since one processor accesses the shared memory continuously. On the other hand, the parking ratio of MP3D is low, since the bus contention is severe because the low cache hit ratio introduces frequent replaces of cache lines.

7 Conclusion

In ATTEMPT-0, the processing speed of the processor is balanced with the speed of the bus, cache memory and the synchronizer. The local memory is useful. Especially, allocating data on the local memory avoids performance degradation due to the write-through protocol. Although the write-through cache is considered to yield poor performance in multiprocessors, fair performance can be achieved by our design policy in which local memory and the synchronizer are used together with the cache. Our evaluation result also shows that bus connected multiprocessors can achieve good performance by appropriately allocating the data according to its nature, without complicated programming methods.

The synchronizer is effectively used in the synchronization with Fetch&Dec operations and exchanges of small sized data. But the amount of the synchronizer memory is too small for general use. Virtual addressing mechanism for the synchronizer should be provided in the next version of ATTEMPT.

The design of the synchronizer is unsatisfactory. It should be more considered about the unit of processing (e.g. processor or process), and the adaptability for process scheduler and virtual memory system. In addition, since we do not have operating system for ATTEMPT-0 which supports the process scheduling yet, performance when the interrupt handling facilities of the synchronizer are used is not evaluated.

Meanwhile, evaluation of the Futurebus shows that starvation is avoided by the fairness release scheme even though the bus location based module address is used for the arbitration. However, it introduces the differences of the bus mastership waiting time and may results an overhead. We believe that only the measurement on the actual machine with practical applications could reveal these problems of the arbitration protocol of Futurebus.

The following problems are also appeared.
Since all the write operation blocks until the operation is completed, write access time for the shared memory is large. Therefore, write operations degrade the total performance. Non-blocking write operations using write buffers should be provided.

Since the DRAMs are designed to be able to be used as either the shared memory or the local memory, the memory system can be optimized for neither of them. As a result, block transfer for the cache replacement is not supported, and the access time of the local memory is large.

Too large size of the processor board introduces the signal propagation delay problem. Thus the designed clock speed can not be achieved.

These problems are solved in the next version of ATTEMPT which is now under implementation.

Acknowledgements

We would like to thank all the members of Amano Laboratory in Keio University. Sunao Torii who has joined NEC corporation, designed and implemented the hardware event monitor board for ATTEMPT-0. Kei Nishino, who is now working at Sony corporation, implemented the Ethernet and the SCSI interfaces. They also helped us to debug the initial implementation of ATTEMPT-0. We would like to express our sincere gratitude to them.

Atsushi Fukumoto who is now working at Toshiba corporation wrote the low level ROM routines and the communication handlers with the host workstation. Jun-ichi Yamato now working at NEC corporation wrote the virtual memory support routines and some system software such as program loader. Junji Yamamoto built the programming environment on the host workstations for ATTEMPT-0.

Finally, our special thanks are due to Toshiyuki Iwata at Allumer corporation for supporting our work with implementation of ATTEMPT-0.

References


Figure 1: ATTEMPT-0

Figure 2: Organization of the synchronizer
Figure 3: Fetch&Dec(X) operation

**Mutex(X) =**

\[
\text{begin} \\
\quad \text{begin} \\
\quad \quad \text{repeat } v \leftarrow \text{Fetch&Dec}(X) \text{ until } v = 1 \\
\quad \quad \{\text{critical region}\} \\
\quad \quad \text{begin} \\
\quad \quad \quad \text{Write}(X,1) \\
\quad \quad \end{begin} \\
\text{end} \\
\text{end}
\]

**Ex.1 Mutual Exclusion**

\{Here, \( n \) is the number of receivers\}

**Sender(X):**

\[
\text{begin} \\
\quad \text{wait until interrupt caused by Fetch&Dec}(X) \text{ of Receivers;} \\
\quad \text{write a message to the region in the main memory;} \\
\quad \text{Write}(X,n) \\
\text{end}
\]

**Receiver(X):**

\[
\text{begin} \\
\quad \text{wait until interrupt caused by Write}(X,n) \text{ of Sender;} \\
\quad \text{read the message from the region in the main memory;} \\
\quad \text{Fetch&Dec}(X) \\
\text{end}
\]

**Ex.2 Synchronization of message multicast**

Figure 4: Examples of the synchronization with the synchronizer
Figure 5: Organization of ATTEMPT-0
Figure 6: Speedup of the applications

Figure 7: Cache hit ratio of the applications

Figure 8: Bus utilization ratio of the applications

Figure 9: The ratio of bus mastership waiting time to the execution time
Figure 10: Precise analysis of the execution time (left) and the bus time (right) of Channel

Figure 11: Precise analysis of the execution time (left) and the bus time (right) of Logique
Figure 12: Precise analysis of the execution time (left) and the bus time (right) of Water

Figure 13: Precise analysis of the execution time (left) and the bus time (right) of Cholesky
Figure 14: Precise analysis of the execution time (left) and the bus time (right) of MP3D.
Figure 15: Execution time of MP3D and Cholesky

Figure 16: Speedup of MP3D and Cholesky

Figure 17: Precise analysis of the execution time of MP3D(AS)
Figure 18: Precise analysis of the execution time of Cholesky(AS)

Figure 19: Precise analysis of the execution time of Cholesky(ULS)

Figure 20: Bus utilization ratio of MP3D and Cholesky

Figure 21: The ratio of the time for write operations to the execution time
Figure 22: Speedup of three implementations of MP3D

Figure 23: The execution time of MP3D and Logique using the synchronizer

Figure 24: Cache hit ratio of MP3D and Cholesky
Figure 25: Bus mastership waiting time: weakest module vs. strongest module

Figure 26: Fairness release ratio

Figure 27: Parking ratio
Table 1: Access time of the memory system

<table>
<thead>
<tr>
<th>Memory System</th>
<th>Access</th>
<th>Access Latency [ns]</th>
<th>Transferred Bytes</th>
<th>Bus Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local Memory</td>
<td>Read/Write</td>
<td>250</td>
<td>4 byte</td>
<td>NO</td>
</tr>
<tr>
<td>Shared Memory</td>
<td>Read hit</td>
<td>150</td>
<td>4 bytes</td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td>Read miss</td>
<td>18200</td>
<td>64 bytes</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>Write</td>
<td>750</td>
<td>4 bytes</td>
<td>YES</td>
</tr>
<tr>
<td>Synchronizer</td>
<td>Read</td>
<td>200</td>
<td>4 bytes</td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td>Write</td>
<td>650</td>
<td>4 bytes</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>Fetch&amp;Dec</td>
<td>750</td>
<td>4 bytes</td>
<td>YES/NO</td>
</tr>
</tbody>
</table>

Table 2: Outline of each application.

<table>
<thead>
<tr>
<th>Application</th>
<th>Contents</th>
<th>Algorithm</th>
<th>Scheduling</th>
<th>Synchronization</th>
<th>Granularity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel</td>
<td>VLSI CAD, channel routing</td>
<td>Hopfield type neural network</td>
<td>static</td>
<td>barrier</td>
<td>large</td>
</tr>
<tr>
<td></td>
<td>Cholesky sparse matrix</td>
<td>supernodal fanout</td>
<td>dynamic</td>
<td>task queue</td>
<td>large</td>
</tr>
<tr>
<td></td>
<td>factorization</td>
<td></td>
<td>(task queue)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logique</td>
<td>logic simulation</td>
<td>query based, Chandy-Misra</td>
<td>dynamic</td>
<td>barrier</td>
<td>small</td>
</tr>
<tr>
<td></td>
<td>Monte Carlo</td>
<td></td>
<td>static</td>
<td>barrier</td>
<td>large</td>
</tr>
<tr>
<td>MP3D</td>
<td>aeronautics, particle-in-cell</td>
<td>Monte Carlo</td>
<td>static</td>
<td>barrier</td>
<td>large</td>
</tr>
<tr>
<td>Water</td>
<td>molecular dynamics, short-range</td>
<td>direct, with cutoff radius</td>
<td>static</td>
<td>barrier</td>
<td>large</td>
</tr>
</tbody>
</table>

Table 3: Propagation delay on Futurebus

<table>
<thead>
<tr>
<th>Kinds of Transfer</th>
<th>Assert</th>
<th>Negate</th>
</tr>
</thead>
<tbody>
<tr>
<td>device and line</td>
<td>26ns</td>
<td>27ns</td>
</tr>
<tr>
<td>filter</td>
<td>82ns</td>
<td>32ns</td>
</tr>
</tbody>
</table>

Table 4: Arbitration time

<table>
<thead>
<tr>
<th>Arbitration</th>
<th>880ns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arbitration with fairness release</td>
<td>1150 ns</td>
</tr>
</tbody>
</table>