Master's Thesis

Title

Proposal and Evaluation of Realization Approach for a Shared Memory System in λ Computing Environment

Supervisor

Professor Masayuki Murata

Author

Hirohisa Nakamoto

February 15th, 2005

Department of Information Networking

Graduate School of Information Science and Technology

Osaka University

Master's Thesis

Proposal and Evaluation of Realization Approach for a Shared Memory System in λ Computing Environment

Hirohisa Nakamoto

Abstract

As users of networks such as the Internet have increased, the amount of traffic has steadily increased. Various applications that utilize images have come to be used, and the demands made on the technology that enables the high speed and large scale transmission in a network have increased. To satisfy these demands, research into optical technologies like IP over a WDM network, GMPLS, and optical packet switches have been studied and developed. However, many such technologies presuppose the existing Internet technology. That is, an IP packet is treated as a degree of granule treating information. As long as architecture based on packet switching technology is focused on, realization of high quality communication to each connection will be very difficult. New technologies such as SAN and Grid computing need to be applied to provide end users with a high speed and reliable communication pipe.

In this thesis, we propose a new computing environment (which we will refer to λ computing environment) that provides a base for parallel computing among nodes distributed in the wide area. In our concept, virtual channels are provided utilizing optical networks connecting computing nodes. It can offer high-speed and reliable connection pipe among nodes, so that it is efficiently applicable to SAN (Storage Area Network) and/or Grid computing. In the environment, shared memory is constituted on a virtual ring of the photonic network. Consequently, it is not necessary to distinguish shared memory in a wide-area distributed system from a communication channel; thus high-speed data exchange between computing nodes on the ring can be achieved. The key to realizing such a computing environment is how to construct a shared memory system on the photonic ring. Thus, we propose and evaluate a shared memory system suitable to the virtual optical ring network, which takes into account contention resolution, cache coherency, and synchronization methods because the propagation delays among nodes are much larger than the conventional shared memory system. Through simulation experiments of using three benchmark programs as representative parallel computing applications, we show the applicability of our shared memory system on the wide-area virtual photonic ring. Our simulation results show that the shared memory and access method for λ computing environment are effective in parallel computation for some of benchmark programs when the number of parallel nodes is less than eight and problem size is large.

Furthermore, we propose and evaluate the method to improve the synchronization between computing nodes to enable collaboration. One of the obtained result is that the method can reduce 40% of the execution clocks when we use one of benchmark program.

Keywords

λ Computing Environment, Photonic Ring Network, Shared Memory, Memory Access Contention, Cache Coherency

Contents

List of Figures

List of Tables

1 Introduction

As users of networks such as the Internet have increased, the amount of traffic has steadily increased. Various applications that utilize images have come to be used, and the demands made on the technology that enables the high speed and large scale transmission in a network have increased. To satisfy these demands, research into optical transmission technology has been actively pursued. Research into WDM technologies that use multiplexed light wavelengths have been the main target for development and technology from new WDM research that can use 1000 wavelengths has also been advanced [1]. In recent years, IP over a WDM network has been studied and developed to provide high-speed transmission on the Internet based on WDM technology. Moreover, standardization of the routing technology of the Internet, called GMPLS, which is the communication technology that uses various optical technologies for a lower layer than the WDM technology, has also been advanced in IETF [2]. Further, aiming to realize the true IP communication of a photonic network, research into optical packet switches based on optical technology has also begun [3-9].

However, many such technologies presuppose the existing Internet technology. That is, an IP packet is treated as a degree of granule treating information, and it is made into the target for research and development of how to carry it at high speed on a network. Therefore, as long as architecture based on packet switching technology is focused on, realization of high quality communication to each connection will be very difficult. New technologies such as SAN and Grid computing need to be applied to provide end users with a high speed and reliable communication pipe; for that, a mass wavelength path needs to be set up between end users and provided for users. That is, it is possible to provide an end user with a ultra high-speed and high quality communication pipe by building a photonic network that uses established fibers, or newly laid fiber if needed, and by utilizing wavelengths multiplexed in the fiber as the minimum particle size for information exchanges.

As middleware aimed at the realization of a high-speed distributed computation environment using an optical network, OptIPuter is proposed [10]. It has been studied and developed to build a Grid environment established on optical networks. It also provides virtual communication paths. However, this is based on present Internet technology and treats a packet as the informational particle size; so that the problem mentioned previously regarding packet processing arises again.

Thus we propose a new architecture, the λ computing environment that has virtual channels utilizing optical fibers connecting computing nodes. In the conventional Grid environment, data is exchanged with the message passing using TCP/IP. In the λ computing environment, by realizing communication between nodes on the Grid not by conventional TCP/IP but by established wavelength paths, we can achieve highly reliable high-speed communication. Then, by making virtual channels on a mesh upon the photonic network that is connected to the network nodes and the computer nodes with optical fibers, distributed computation on a high speed channel is enabled. Moreover, it is possible to utilize wavelengths as a shared memory by constituting a virtual ring in the λ computing environment [11]. As a result, it is not necessary to distinguish shared memory from communication channels in a wide-area distributed system; and we expect that the high-speed data exchange between computing nodes can be achieved (see Fig. 1). In our research group, we also use wavelengths as a high-speed transmission channel and implemented high-speed access method to the shared memory that exists on each computing node [12].

In this thesis, we propose and evaluate an approach to realize a shared memory system using a virtual optical ring network. Specifically our shared memory system uses a level-1 cache in a CPU of each computer group as the cache of such a shared memory. When using the virtual ring as a shared memory, it is necessary, unlike a bus between a CPU and the shared memory in a computer, to consider restrictions in timing and the frequency of access, since the shared memory is spread out on a long-distance optical fiber; so to take into consideration coherency between the shared memory in the virtual ring and the cache of each computer group more strictly than the conventional shared memory system. Next we have to solve the contention of shared memory access like a conventional shared memory system. This problem arises in cases where a processor has not finished writing access to the data on the shared memory, and another processor tries to read or write to the same data. When we perform parallel computation on shared memory using a virtual optical ring network, synchronization is needed to collaborate between computing nodes. However, it is not necessary to distinguish the shared memory in a wide-area distributed system from a communication channel so that it appears that the high-speed data exchange between computing nodes is achieved. As noted above, after considering such features, we propose a shared memory access method for the λ computing environment, and evaluate the method through simulations.

The rest of the thesis is organized as follows. In Section 2, we describe the cache coherency problem, memory access contention, and synchronization between computing nodes of the con-

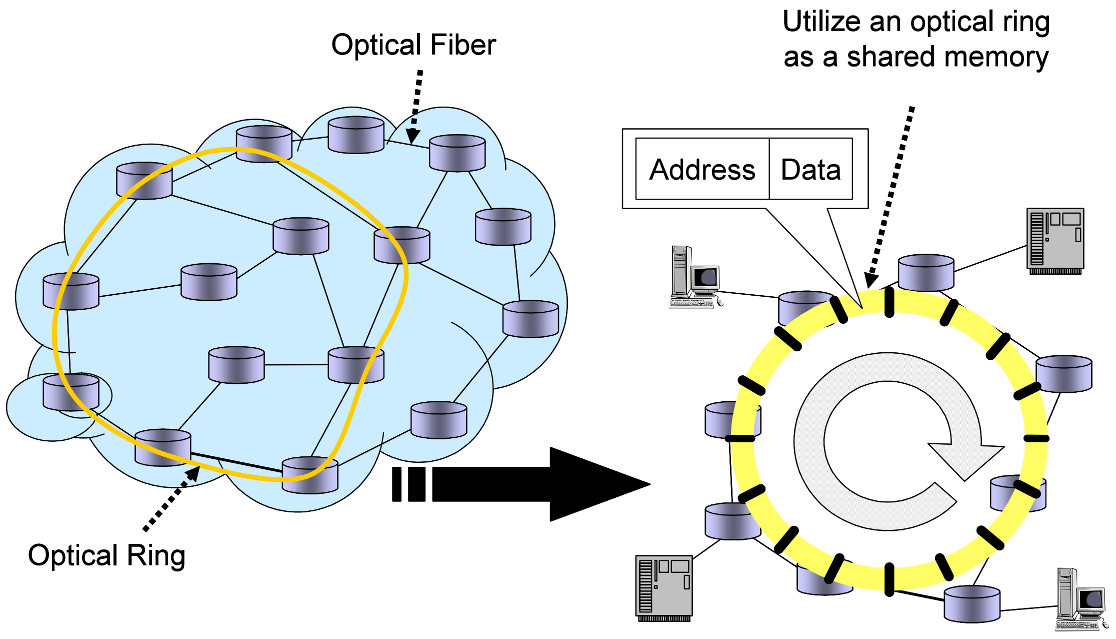

Figure 1: λ Computing Environment.

ventional shared memory system. In Section 3, we propose a realization approach for a shared memory system in the λ computing environment. In Section 4, we evaluate our approach using a benchmark program for parallel computing. In Section 5, we explain how to improve the shared memory access method. Finally, we conclude the thesis and describe future work in Section 6.

2 Conventional Shared Memory System

To realize a shared memory system, it is necessary to avoid memory access contention, maintain cache coherency [13, 14], and realize synchronization between computing nodes to enable collaboration. In this section, we briefly explain these techniques.

2.1 Contention Avoidance in a Shared Memory System

A problem exists when a processor has not finished writing access to data on the shared memory, and another processor tries to read or write to the same data. This kind of contention is solved by a lock mechanism. Each processor has to lock the shared memory before it tries to write the data on the shared memory (Fig. 2). By using a lock mechanism, write access to the shared memory is protected. When we realize a shared memory system in the λ computing environment, we have to resolve this problem.

2.2 Cache Coherency

When each processor has a cache, it is necessary to fully take into consideration the consistency between the data on the cache and the data on the shared memory (Fig. 3). Two ways, a directory method and a snoop cache method, are techniques for generally maintaining cache coherency. In realizing shared memory on the λ computing environment, access to a directory table may become a bottleneck when a directory method is adopted. So a snoop cache method is adopted in this thesis.

The snoop cache method offers cache consistency between data on caches and on the shared memory. It snoops memory access on a share bus, and performs consistency control to a local cache block by the distributed technique if needed. When a processor tries to read data, firstly it searches in the local cache, and when data does not exist there, it accesses to the shared memory. When a processor tries to read or write to a shared memory, consistency control is not needed if another processor does not the same data in its local cache. However, if another processor has the same data in its local cache, some methods can be considered using a method to keep cache consistency. Moreover, when a processor writes or updates the data on its cache, keeping cache consistency becomes still more complicated and there are some ways to keep consistency.

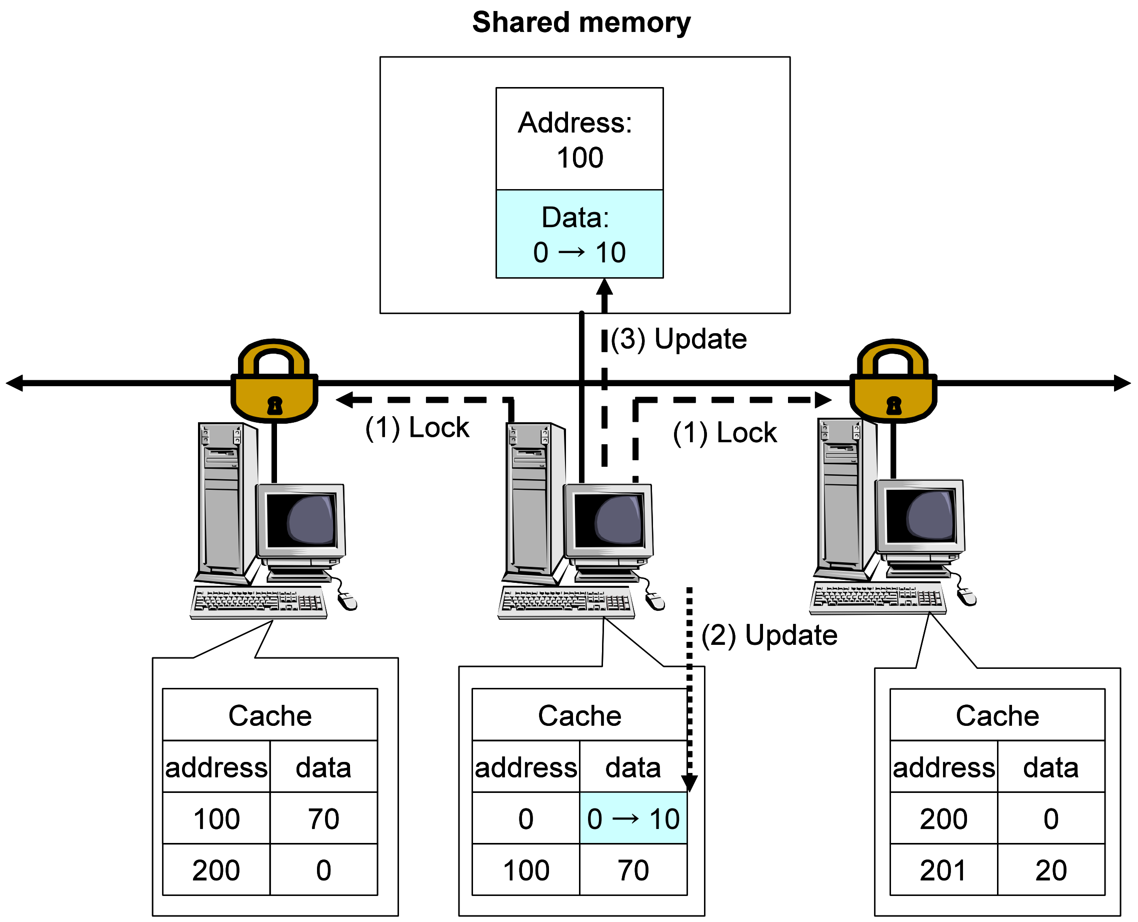

Figure 2: Lock Mechanism.

Such cache consistency protocols are classified into four types according to the timing (write-through, write-back) and the method (invalidation, updating). Among those, a write-back invalidation protocol has the least shared memory access. When the access delay time to a shared memory is large, this protocol is very effective. So we adopt this protocol when we realize a shared memory system in the λ computing environment. The write-back invalidation protocol is simply explained below.

In the write-back invalidation protocol, data in the local cache has three states; Invalid (I), Clean (C), and Dirty (D). The I state means that data is invalidated and cannot be used, the C state means that the data on the cache is the same value compared to the data on the shared memory, and the D state means that the data on cache is not the same value compared to the data on shared memory. We show the state transition diagram of the basic write-back invalidation protocol in

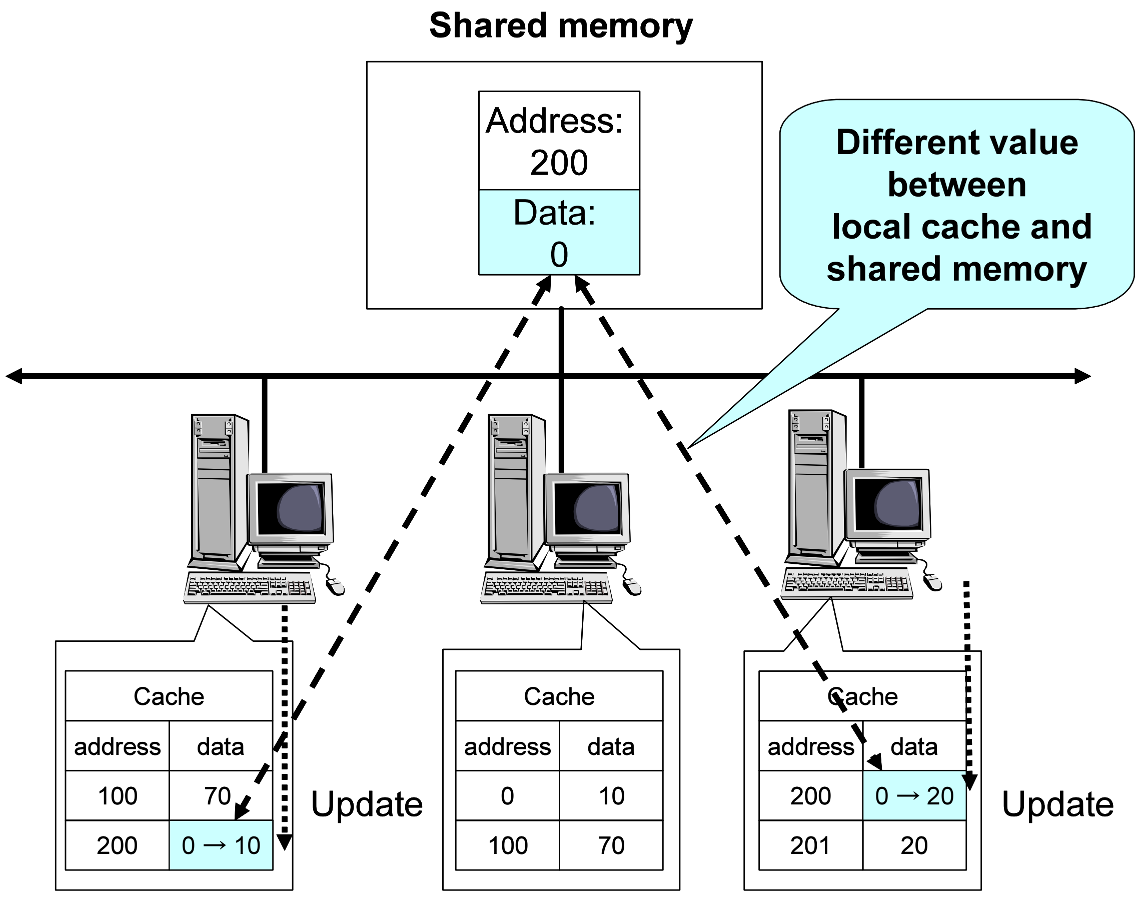

Figure 3: Cache Coherency Problem.

Fig. 4.

When one or some computers refer to an address, the data is copied to A cache from the shared memory and that cache's state becomes a clean state as shown in Fig. 5 (a). Since the value of the data on the shared memory and the data in the C state is the same, read access to this cache block does not need bus operation. If a processor writes to data in a C state, the state will become a D state. At this time, the control message requesting invalidation of the relevant data is sent on a bus. Since the cache controllers of other processors snoop a bus, they receive the control message and invalidate the relevant data in their local cache (Fig. 5 (b)). Henceforth, read and write accesses to the data in a D state do not need bus operation. When other processors read to the data in a D state, the data in a D state is written back to the shared memory, and cache consistency is completed. Next, the data is sent by bus to the processor that requested the read access and states of cache blocks on both processor's cache will become a C state (Fig. 5 (c1)). On the other hand, when

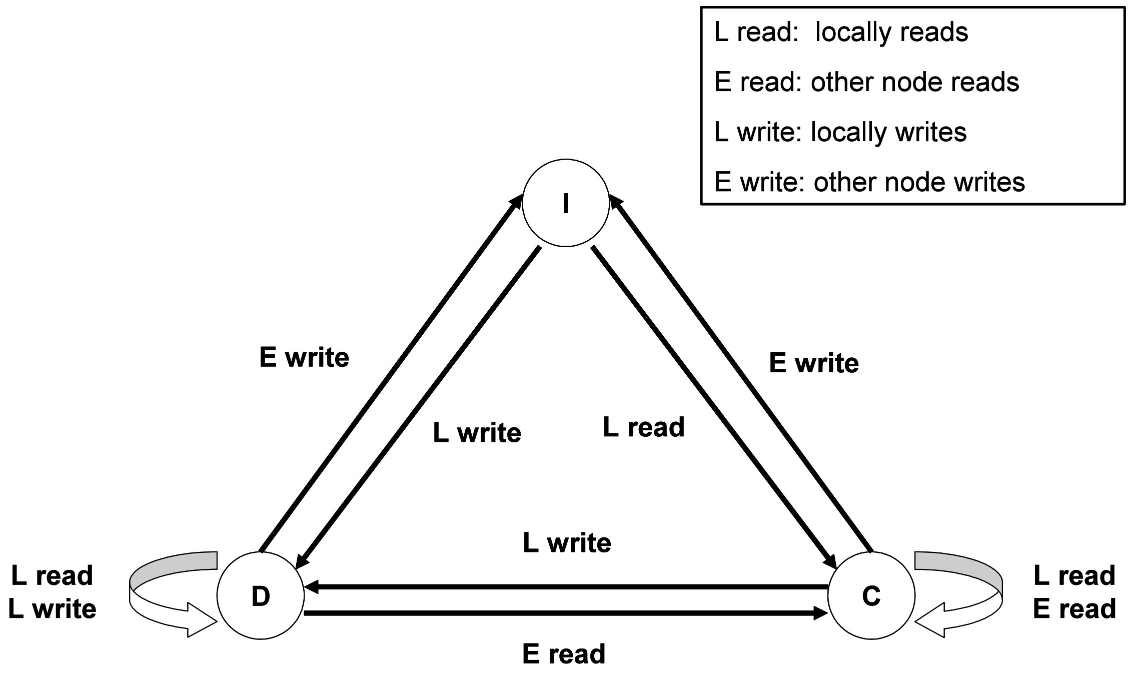

Figure 4: State Transition Diagram.

other processors write to the address of the data in a D state, like reading, writing back of the data in a D state to the shared memory takes place, and the data is sent to the processor which sent the demand message. Finally, the demanding processor writes the data on a local cache, and the state of the data will be D. The cache data on the processor that has the original data are invalidated (Fig. 5 (c2)).

2.3 Synchronization between Computing Nodes

When we perform application programs of parallel computation on a shared memory system, synchronization between computing nodes is needed to collaborate. Kinds of synchronization include atomic operation, caching of synchronous variable, blocking control on a shared memory, a memory lock, and the barrier synchronization method. Barrier synchronization is used when each processor needs to wait until all processors reach to the same break point.

Application programs used for evaluation in this thesis first calculate locally and then after local calculation perform synchronization processing. In barrier synchronization, each computing node reads and updates the data of synchronous variable by turns. So a synchronous memory on an optical ring is suited to barrier synchronization because data is circling on the optical ring and

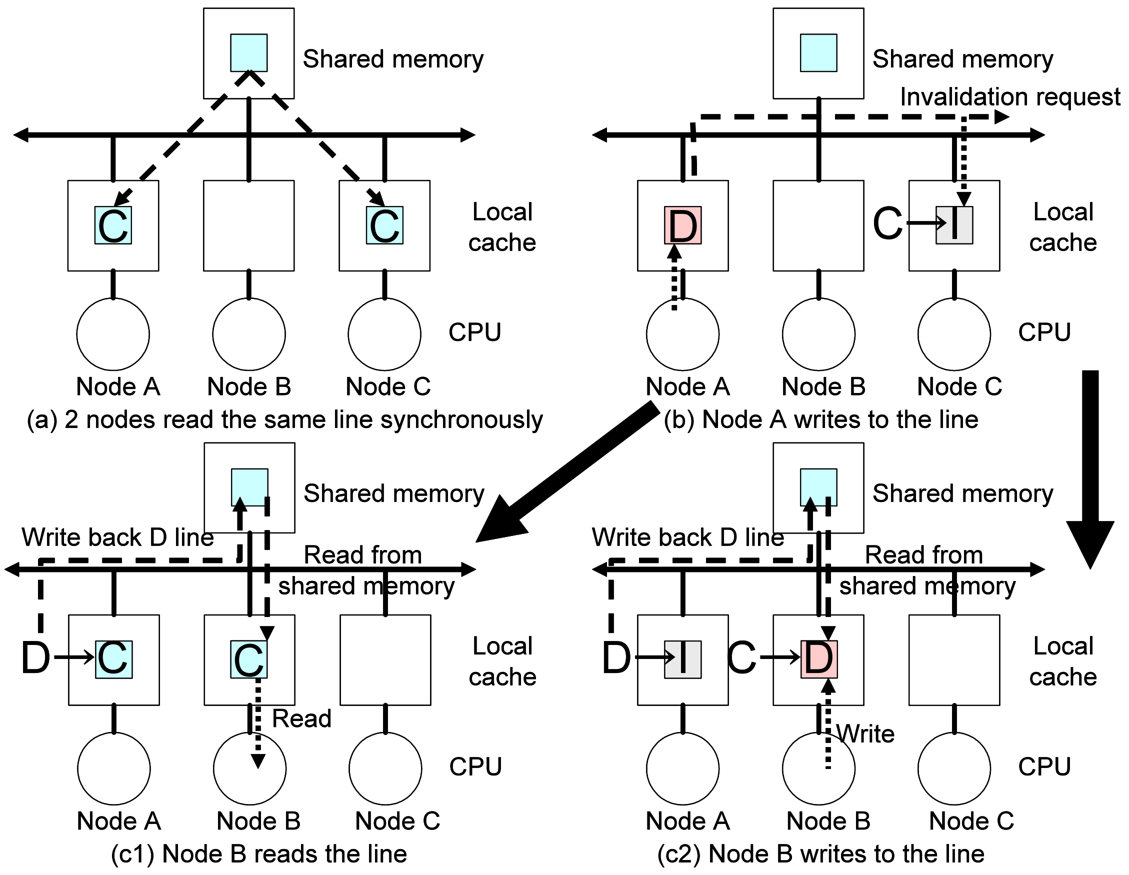

Figure 5: Behavior of Write-back Invalidation Protocol.

each computing node can easily read and update the data on a synchronous memory by turns. So in this thesis we adopt the barrier synchronization method. The method of realizing barrier synchronization using shared memory on an optical ring is shown in Sec. 3.4.

3 Realization Approach for a Shared Memory System in the λ Computing Environment

In this section, we explain an approach for realizing a shared memory system in the λ computing environment. Firstly, we describe our network model. Like a conventional shared memory system, we have to realize cache coherency, avoidance of memory access contention, and synchronization between computing nodes to enable collaboration. We also describe how to resolve these points needing consideration.

3.1 Network Model

We show a network model in Fig. 6. Computing nodes that compose the λ computing environment are connected with optical fibers that make a virtual ring network. In this thesis, we presuppose that each computing node has one CPU, a level-1 cache, and a local memory. A local memory is used for storage of programming codes and local data, a shared memory is used for storage of the shared data that all computing nodes use in computing and a synchronous memory is used for synchronization between computing nodes. An optical ring network has a wavelength path for shared memory, a wavelength path for control signals and a wavelength path for synchronous memory. The bandwidth of an optical ring for shared memory is set to 1Tbps, optical ring length is 10km, and the number of optical rings for shared memory is one. Since propagation delay time is 5 ns/m, we can use an optical ring network as a shared memory, of which the capacity is equivalent to 6.25MBytes. We usually use these parameters except otherwise explicitly stated. For comparative evaluation, we also use the model with 1km ring length and 10 number of rings in parallel, and with 100m ring length and 100 number of rings while these models might be unrealistic even in the near future in Sec. 4.3. The processing delay time in the interface of each computing node and middle nodes, such as network devices which constitute an optical ring network, is not explicitly taken into account here. Indeed, we assume that it is included in the propagation delay time.

Next, we show a configuration of each computing node and data flows between a computing node and wavelengths of an optical ring in Fig. 7. The local cache controller searches the data on the local cache when it receives read or write request to the data on the shared memory and the synchronous memory from the CPU. If the data requested by the CPU is not found in the local cache, the local cache controller commits the request from the CPU to the shared cache controller.

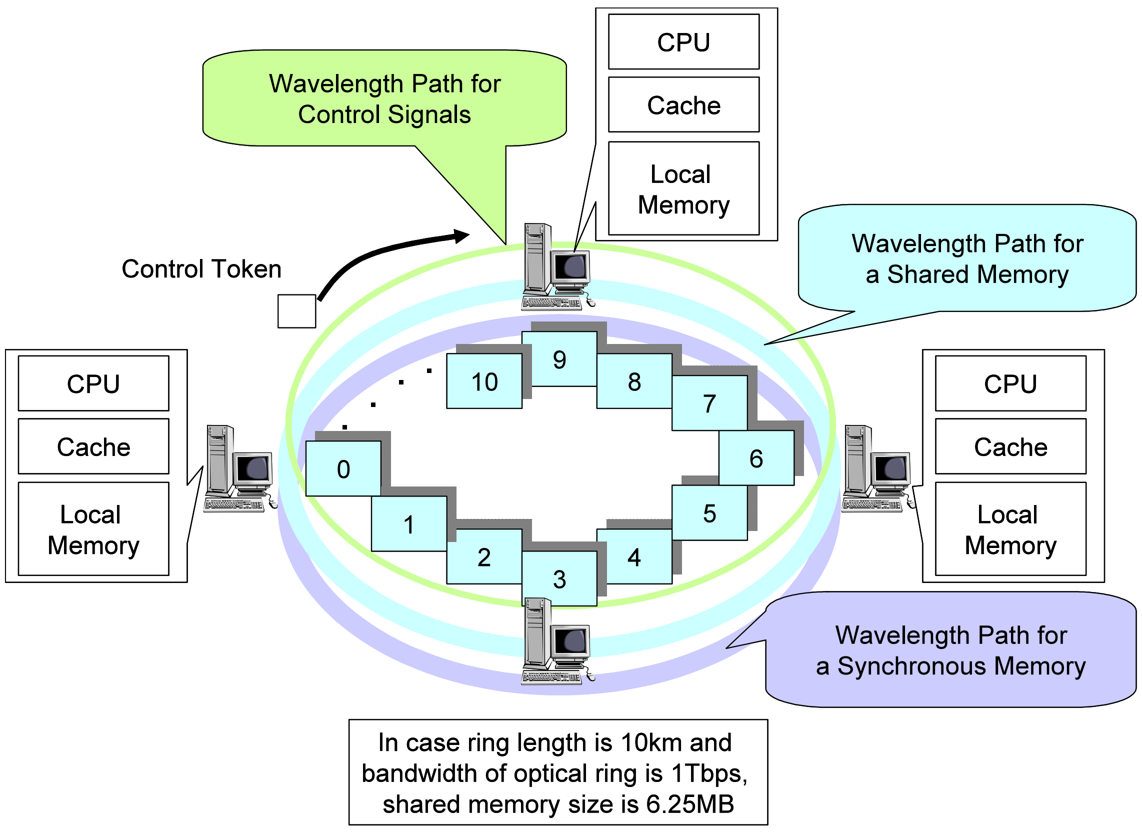
Wavelength Path for Control Signals
CPU
Cache
Local Memory
Control Token
Wavelength Path for a Shared Memory
CPU
Cache
Local Memory
CPU
Cache
Local Memory
0
1
2
3
4
5
6
7
8
9
10
Wavelength Path for a Synchronous Memory
In case ring length is 10km and bandwidth of optical ring is 1Tbps, shared memory size is 6.25MB

Figure 6: Network Model.

The shared cache controller monitors wavelengths for control signals, a shared memory, and a synchronous memory. When it receives the request from the local cache controller, it reads or updates the data on the shared memory and the synchronous memory. And when it receives the control messages from other nodes, it changes the state of the cache block on a local cache to keep cache coherency.

3.2 Contention Avoidance

Like a conventional shared memory system, we have to solve contention to the shared and synchronous memories to realize a shared memory system in the λ computing environment. To resolve this problem, we adopt a lock mechanism. That is, each computing node has to send a lock request message using a control token before it tries to write or update data in state C on a local cache, and data on the shared memory or the synchronous memory. By using this mechanism, write access to the shared memory is protected. The approach to realize contention avoidance is

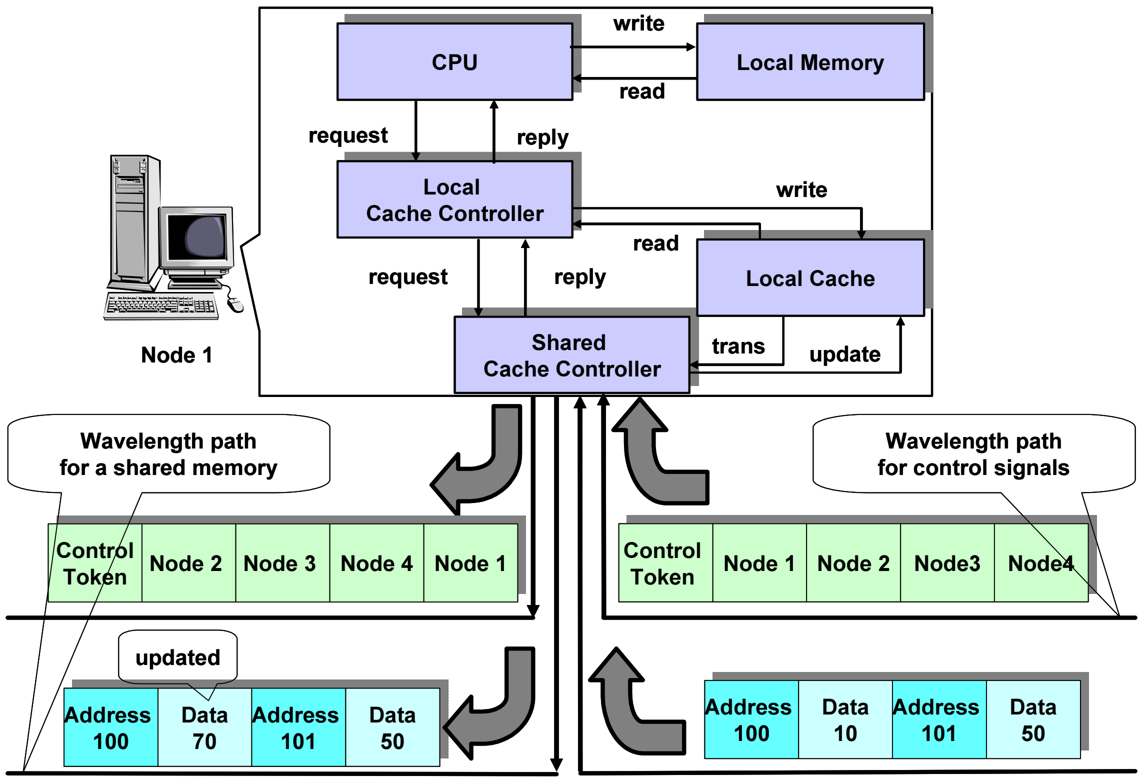

Figure 7: Cache Control Scheme and Data Flows.

explained in Sec. 3.3 along with cache coherency.

3.3 Cache Coherency

Like a conventional shared memory system, we have to realize cache coherency to realize a shared memory system in the λ computing environment (Fig. 8). As mentioned in Sec. 2.2, the write-back invalidation protocol has the least shared memory access. It is very effective because in our shared memory system, the access delay time to a shared memory is large. Thus, there are some write-back invalidation protocols; such as the Illinois [15] and Symmetry protocols [16]. We adopt the Illinois protocol because its consistency control is the simplest. In the Illinois protocol, data of a local cache has four states; Invalid (I), Clean Exclusive (CE), Clean Shared (CS) and Dirty (D). I and D states are the same states like in the conventional protocol. The CE state means that the data on the cache is the same value compared to the data on the shared memory and another processor does not have the same data, and the CS state means that another processor has the same data in its local cache. However, the Illinois protocol presupposes the shared memory system using a

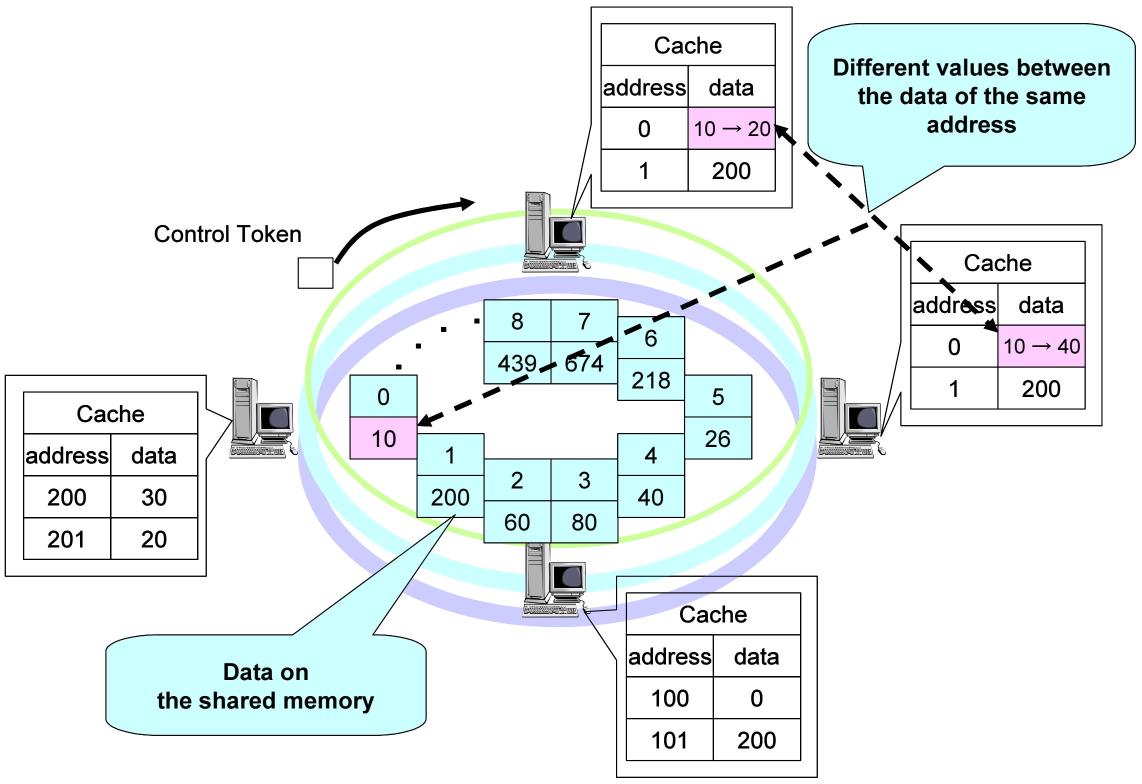

Figure 8: Cache Coherency Problem in λ computing environment.

shared bus. So we have to adapt Illinois protocol to the shared memory system in the λ computing environment.

Accordingly we propose a new cache coherency protocol that solves cache coherency and contention to the shared and synchronous memories in the λ computing environment on the basis of the Illinois protocol. We show the state transition diagram of our proposed cache coherency protocol in Fig. 9. Hereafter, we first explain each arrowhead of the transition in Fig. 9. Second, we explain the behavior of the computing nodes that received a control message. In our cache coherency protocol, five control messages are used. A line copy request message is sent by a computing node that tries to read the data when it does not have the data in its local cache. A line move request message is sent by a computing node that tries to write the data when it does not have the data in its local cache. Lock and invalidation request messages are sent by a computing node that tries to update the data in the CE or CS states or write the data in a I state. A lock request message is used to avoid contention to shared and synchronous memories. An invalidation request message is used to keep that the cache block in the D state is only one among all computing nodes

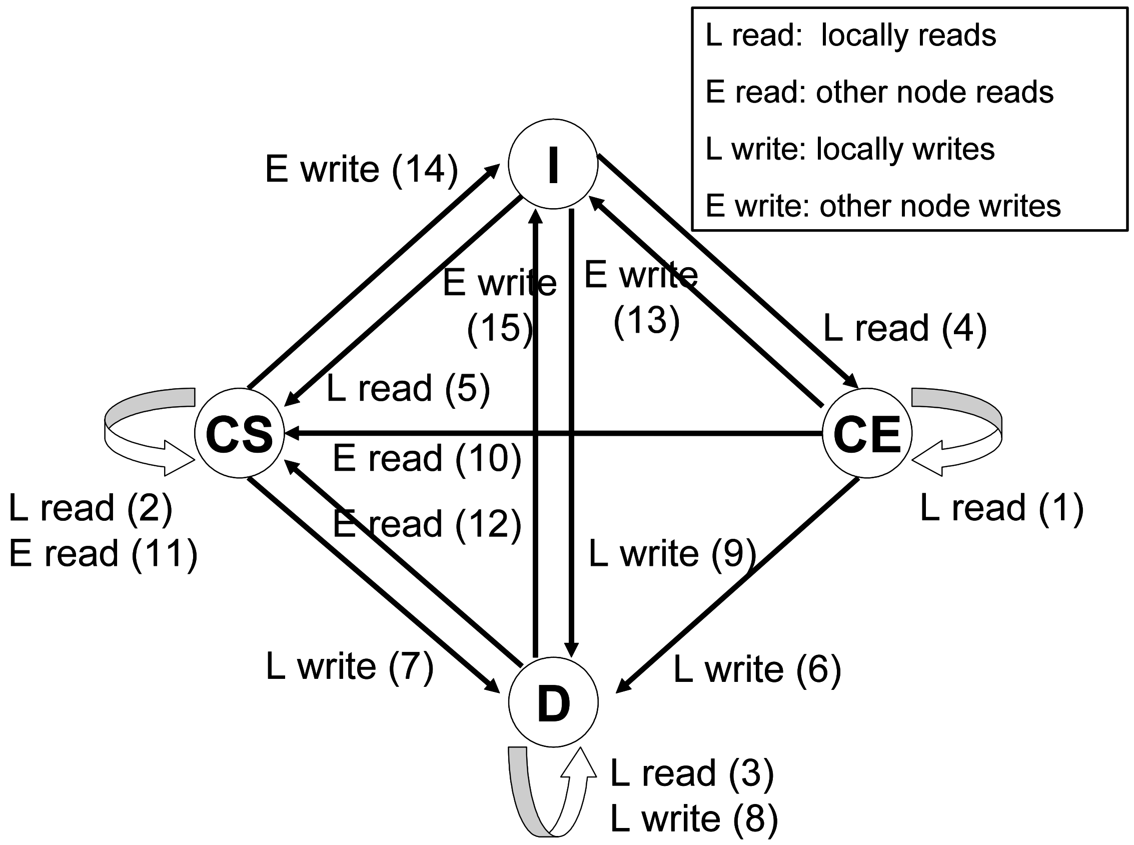

Figure 9: State Transition Diagram.

for maintaining cache coherency. An unlock request message is sent by a computing node that has finished updating or writing the data.

At first, we explain the case a computing node that reads the data from A local cache. In this case, the state of the cache block (Fig. 9 (1), (2), (3)) does not change.

We next describe how a computing node reads data from another computing node's cache or shared memory. A computing node attaches a line copy request message to a control token after having the control token. It waits until the control token returns. If it receives a line copy ack (Fig. 9 (5)), it has to wait until the copy of the cache block is sent by another computing node and read the cache block from the control token. Then it changes the state of the cache block ($I \rightarrow CS$). If it does not receive a line copy ack (Fig. 9 (4)), it directly reads the cache block from the shared memory. Then it changes the state of the cache block in its local cache ($I \rightarrow CE$).

Next, we explain how a computing node updates the data on a local cache. If the state of the cache block is D (Fig. 9 (8)), it only updates the data on the local cache. If the state of the cache block is CE (Fig. 9 (6)), it updates the data on the local cache and changes the state of the cache block ($CE \rightarrow D$). If the state of cache block is CS (Fig. 9 (7)), contention avoidance processing is needed. After having the control token, it has to check whether a lock request message of another computing node is attached to the control token. If a lock request message is attached, it has to wait until an unlock request message is sent and then restart the write processing. If a lock request message is not attached, it attaches a lock request message and an invalidation request message to the control token. Then it updates the data on the local cache and changes the state of the cache block ($CS \rightarrow D$). Finally, it attaches an unlock request message to the control token after having the control token.

It is then illustrated how a computing node tries to write when it does not have the copy cache block on the local cache (Fig. 9 (9)). In this case, contention avoidance processing is also needed. A computing node searches the lock table. If the address of the cache block that it tries to write is registered in the lock table, it has to wait until an unlock request message is sent. If the address is not registered, it has to check whether a lock request message of other computing node is attached to the control token after having the control token. If a lock request message is attached, it has to wait until an unlock request message is sent and then restart the write processing. If a lock request message is not attached, it attaches a lock request message and a line move request message to the control token. It waits until the control token returns. If it receives a line copy ack, it has to wait until a cache block is sent by another computing node and read the cache block from the control token. Then it updates the cache block and changes the state of the cache block ($I \rightarrow D$). If it does not receive a line copy ack, it directly reads the cache block from the shared memory. Then it updates the cache block and changes the state of the cache block ($I \rightarrow D$). After updating the cache block, it attaches an unlock request message to the control token after having the control token.

the behavior of computing nodes that received a control message is as follows. At first, we explain the behavior of a computing node that receives a line copy request message. If a computing node has a cache block of the relevant address in the CE state (Fig. 9 (10)), it attaches a line copy back and a copy of cache block to the control token after having the control token. Then, it changes the state of the cache block ($CE \rightarrow CS$). If a computing node has a cache block of the relevant address in the CS state (Fig. 9 (11)), it attaches a line copy back and a copy of the cache block to the control token after having the control token. If a computing node has a cache block of the relevant address in the D state (Fig. 9 (12)), it attaches a line copy back to the control token after having the control token. Then, it writes back the cache block in the D state to the shared memory and changes the state of the cache ($D \rightarrow CS$). It attaches a copy of the cache block to the control token after having the control token.

A computing node that receives a line move request message takes a following action. If a computing node has a cache block of the relevant address in the CE state (Fig. 9 (13)), it attaches a line move ack and copy of the cache block to the control token after having the control token. Then, it changes the state of the cache block ($CE \rightarrow I$). If a computing node has a cache block of the relevant address in the CS state (Fig. 9 (14)), it attaches a line move ack and a copy of the cache block to the control token after having the control token. Then, it changes the state of the cache block ($CS \rightarrow I$). If a computing node has a cache block of the relevant address in D the state (Fig. 9 (15)), it attaches a line move ack to the control token after having the control token. Then, it writes back the cache block in the D state to the shared memory and changes the state of the cache block ($D \rightarrow I$). It attaches a copy of the cache block to the control token after having the control token.

We finally explain the behavior of a computing node that receives lock, unlock and invalidation request messages. If a computing node receives a lock request message, it registers the relevant address to its lock table. If a computing node receives an unlock request message, it removes the relevant address from its lock table. If a computing node receives an invalidation request message, it searches the relevant address in the local cache. If a computing node has a relevant cache block in its local cache, it changes the state of the cache block to the I state.

3.4 Synchronization between Computing Nodes

We explain the method for realizing barrier synchronization in the shared memory system using an optical ring network. First, part of the wavelength paths of an optical ring are allocated to the synchronous memory area. When a synchronous memory is accessed, a Fetch $\&$ Decrement operation like in the conventional method is indivisibly performed. That is, it ensures that access to a synchronous memory indivisibly causes a subtraction processing of the relevant data. Since only one computing node can simultaneously access the synchronous memory, when using an optical ring network for a synchronous memory, execution of an atomic operation is easy. With an application program, in bringing about synchronization among some computing nodes, each node

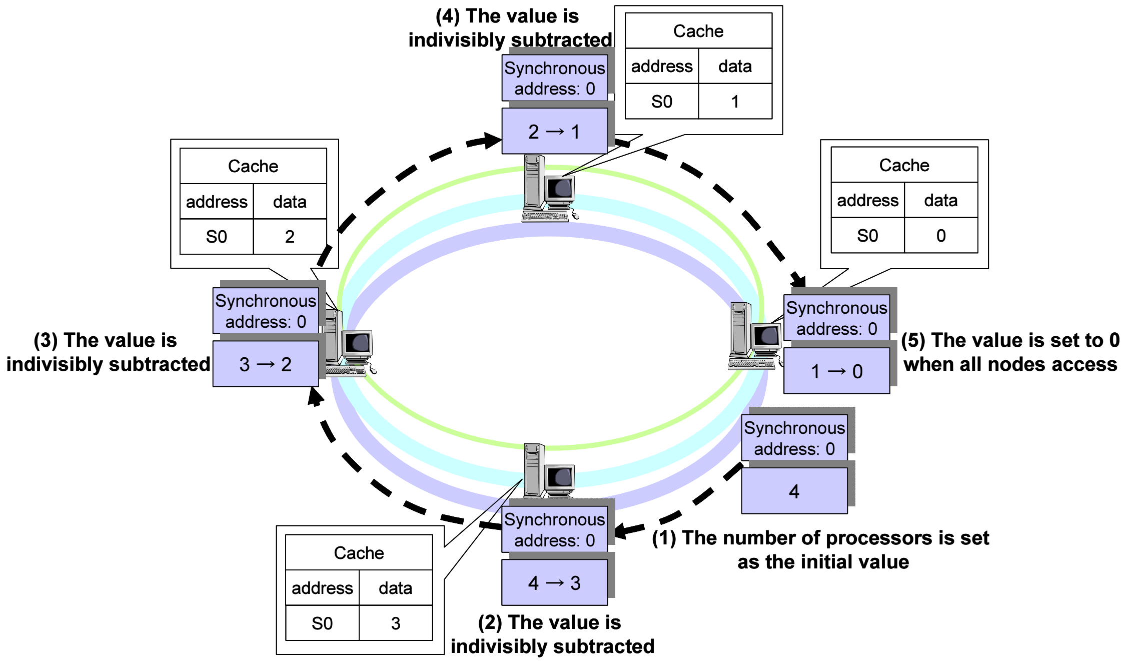
(4) The value is indivisibly subtracted
Cache
address
data
S0
1
Synchronous address: 0
2 → 1
Cache
address
data
S0
2
Cache
address
data
S0
0
Synchronous address: 0
(3) The value is indivisibly subtracted
3 → 2
Synchronous address: 0
1 → 0
(5) The value is set to 0 when all nodes access
Synchronous address: 0
4
Synchronous address: 0
(1) The number of processors is set as the initial value
Cache
address
data
S0
3
4 → 3
(2) The value is indivisibly subtracted

Figure 10: Barrier Synchronization in λ computing environment.

accesses to the synchronous memory. We show our approach for realizing barrier synchronization in λ computing environment in Fig. 10. The number of processors is set in the synchronous memory as the initial value (Fig. 10 (1)). Each computing node accesses to the synchronous memory by turns (Fig. 10 (2), (3), (4)). The value of the synchronous memory will be set to zero if all nodes access (Fig. 10 (5)). If the value of a synchronous memory is set to zero, all nodes finish the synchronous process and begin the next processing.

4 Performance Evaluation

In this section, we evaluate through simulation the performance of the shared memory access method proposed in the previous section. We utilized the ISIS library [17, 18] currently developed at the Amano Laboratory of Keio University in coding the simulation program.

4.1 Simulation Model

We used the following network model. Each computing node in the λ computing environment is interconnected with optical fibers and nodes are configured to virtually form the ring topology. Each computing node has one CPU, a level-1 cache, and a local memory. Clock frequency of a CPU is 1GHz, the capacity of a level-1 cache is 512KB, and the capacity of a local memory is 2GByte. It assumes that computing nodes are put on the optical ring network with equal distances. An optical ring network has wavelength paths for shared memory, control signals, and synchronous memory. The bandwidth of an optical ring for shared memory is set to 1Tps, optical ring length is 10km, and the number of optical rings for shared memory is one. Since propagation delay time is 5 ns/m, we can use an optical ring network as a shared memory, of which the capacity is equivalent to 6.25MBytes. We usually use these parameters except otherwise explicitly stated. For comparative evaluation, we also use the model with 1km ring length and 10 number of rings in parallel, and with 100m ring length and 100 number of rings while these models might be unrealistic even in the near future. The processing delay time in the interface of each computing node and middle nodes, such as network devices which constitute an optical ring network, is not explicitly taken into account here. Indeed, we assume that it is included in the propagation delay time.

To evaluate the performance, we use a Splash2 benchmark program [19], such as the "radix sort" program that sorts the sequence of an integer value using a radix sort algorithm. We also use the "product of a matrix" program that calculates the product of $n \times n$ matrix and the "queen problem" program that solves the n-queen problem. See Table 1 for comparing the characteristic of memory accesses for these programs. The numbers show the order of frequencies in memory accesses of programs. See also Figs. 11 and 12 for actual data.

We note that sample programs that we are using in this thesis are just intended to see how levels of parallelism affects the performance. In actual, the program could be clearly divided

Table 1: Characteristics of Memory Access (Ring Length 10km).

	nodes	clocks	shared accesses	synchronous accesses	cache hit ratio
radix sort (16384 keys)	1	1,239,504,563	116,224	0	59.96%
	2	1,079,750,949	1,296,305	55	17.24%
	4	772,450,953	4,822,926	131	16.68%
	8	670,200,953	17,821,287	379	16.76%
	16	865,300,953	71,520,173	1,259	16.86%
product of a matrix (128×128)	1	2,048,357,893	81,920	0	60.24%
	2	2,665,750,970	162,124,605	13	79.54%
	4	2,615,251,057	471,528,629	21	90.03%
	8	2,539,751,057	1,086,505,926	37	95.25%
	16	2,506,851,057	2,316,505,719	69	97.54%
queen problem (32×32)	1	110,045,031	499,250	0	30.48%
	2	2,494,650,912	8,964,784	619	15.64%
	4	2,325,700,916	8,869,410	1,031	17.04%
	8	2,074,550,916	20,263,999	1,855	15.93%
	16	2,107,500,916	35,978,138	3,503	17.65%

into independent tasks when applied to parallel computation, and we can enjoy a parallelism of computation by an increasing number of nodes, but some part of the entire program needs synchronization to an extent, and to effect on the performance depends on the problem. The three programs that we have chosen here are typical examples and have different characteristics as indicated in Table 1. Our intention here is to test whether the typical parallel algorithm can work well in a sense that total parallel execution time is not unacceptably increased. From this perspective, our result indicates that an introduction of parallelism does not only result in the increasing execution time.

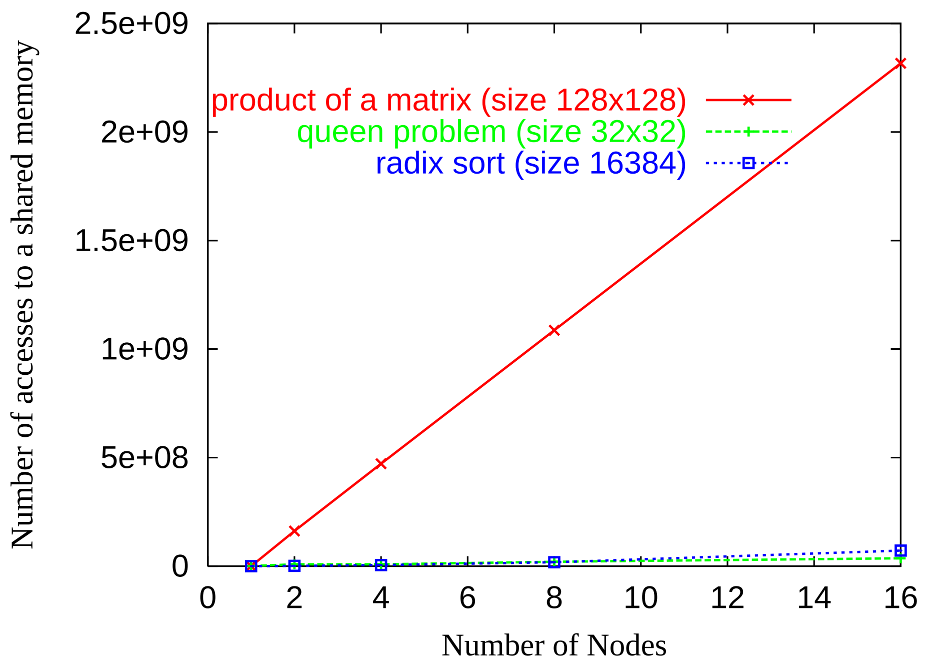

Figure 11: Number of Accesses to a Shared Memory (Ring Length 10km).

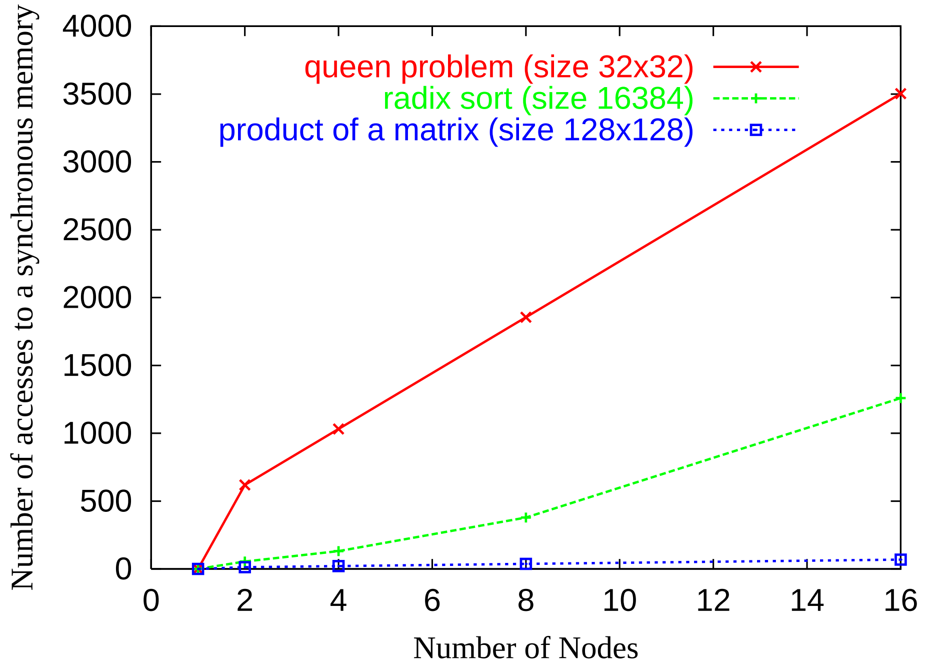

Figure 12: Number of Accesses to a Synchronous Memory (Ring Length 10km).

4.2 Comparisons: Basic Results

We show results of execution time for each application program by setting the ring length to 10km. The number of execution clocks in CPU for the “radix sort” program is first shown in Fig. 13. The numbers of keys for sorting are set at 4096, 8192, and 16384. When the number of key is 4096, the advantage of parallel computation cannot be observed even if the number of computing nodes is increased. This is because the ratio of synchronous operation to total operation is large. However, as problem size becomes larger such as 8192, it turns out that the advantage of parallel computation appears. As the number of nodes exceeds some number (4 in the case that the key size is 8192), the execution time is gradually increased because the number of synchronization becomes large by an increasing number of nodes. When the sort key size is 16384, such a tendency becomes clearer. From these results, we found that the shared memory and access method for λ computing environment are effective in parallel computation for the “radix sort” program when the number of parallel nodes is not so large.

The case of “product of a matrix” program is next shown in Fig. 14. The matrix sizes are changed from 32×32 to 128×128. The advantage of parallel computation becomes smaller in this case. It is because the number of accesses to the shared memory is large compared to other application programs as shown in Fig. 11, where the numbers of accesses to the shared memory are compared in three programs, and it compensates for introduction of parallel computation. However, it is still true that it does not introduce the additional delay if the “product of a matrix program” does not occupy the large portion of the entire program.

The same tendency can be observed in Fig. 15 where the “queen problem” program is considered. Its problem size is changed from 8×8 to 32×32. In this case the number of synchronous accesses is much larger than other programs as shown in Fig. 12, where the numbers of accesses to the synchronous memory are compared in three programs. However, we can again see that the execution time is at least not increased even if the number of nodes becomes large.

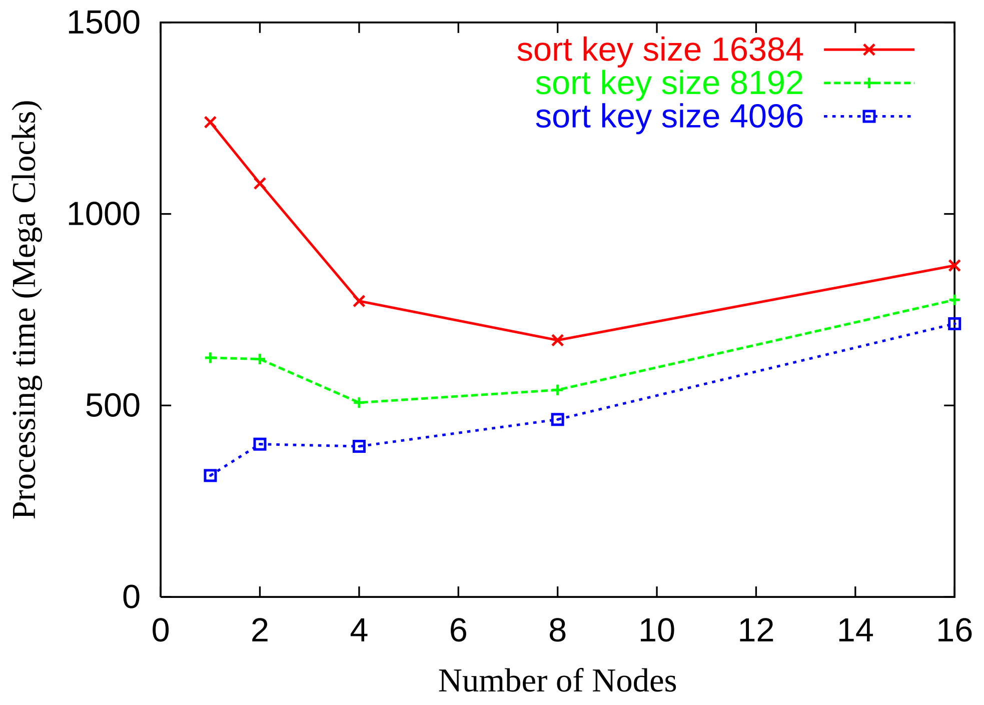

Figure 13: Processing Time of “radix sort” Program (Ring Length 10km).

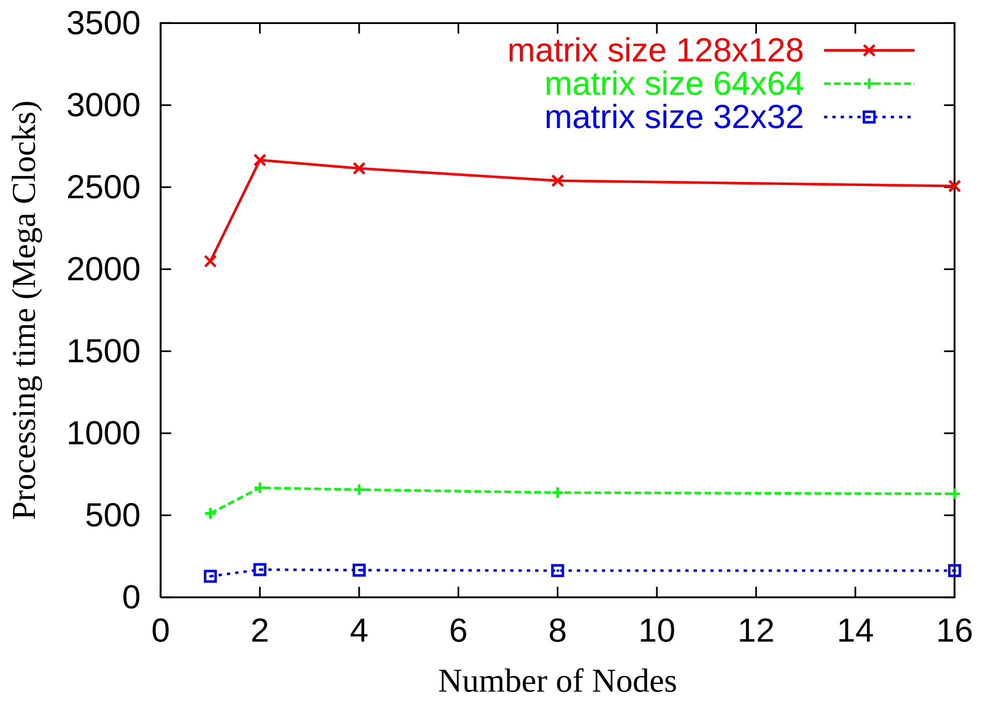

Figure 14: Processing Time of “product of a matrix” Program (Ring Length 10km).

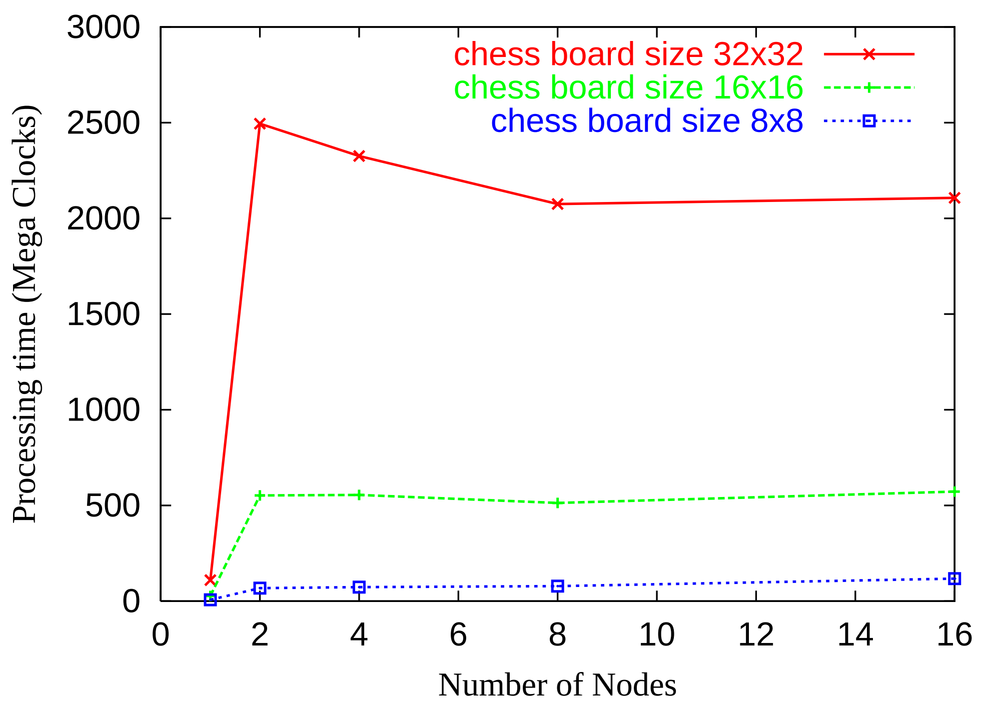

Figure 15: Processing Time of “queen problem” Program (Ring Length 10km).

4.3 Effect of Increasing the Optical Ring Length for Parallel Computation

When ring length is 10km long, the access delay time to the shared memory and the synchronous memory are large and may be the main factor that compensates for introduction of parallel computation. And when ring length becomes shorter, the advantage of parallel computation may become clearer. Accordingly, we next investigate the effect of decreasing the ring length on parallel computation time. We use three values of ring lengths: 100m, 1km and 10km.

In Figs. 16 and 17, the execution times of the “radix sort” program are shown against 1km and 100m of ring length. Compare with Fig. 13 where the case of 10km ring length was shown. We can here see that the results are almost same in three cases.

A different behavior is observed in using the “product of a matrix” program. See Figs. 14, 18, and 19. When the ring length was 10km (Fig. 14), there is no effect of parallel computation due to the access delay time is large much larger than the shared memory time. However, when the ring length is 1km (Fig. 18) and the matrix size is large enough (256×256), the effect clearly appears when the number of nodes is less than eight. It is because the hit ratio of the level-1 cache becomes high as the matrix size becomes large. Also, when the ring length is 100m (Fig. 19), the effect of parallelism is attained even if the matrix size is small (64×64 or 128×128). As we have already mentioned, the “product of a matrix program” has a characteristic that the number of accesses to the shared memory is the largest among all the application programs. See Fig. 11. Then when ring length is short, the effect of parallel computation is obtained. Moreover, the number of accesses to the synchronous memory is the smallest, and therefore, parallel computation can be easily improved as the number of nodes increases.

In the “queen problem” program, on the other hand, no advantage of parallel computation is obtained even if the ring length is changed as shown in Figs. 15, 20, and 21. It is due to the largest number of synchronous accesses in three programs even though the accesses to the shared memory do not frequently occur.

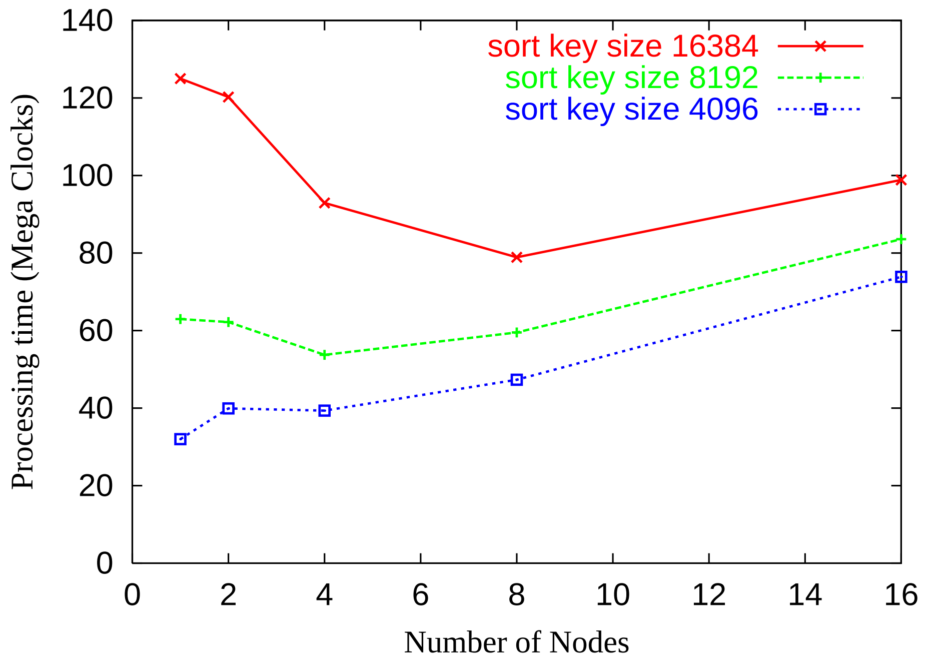

Figure 16: Processing Time of “radix sort” Program (Ring Length 1km).

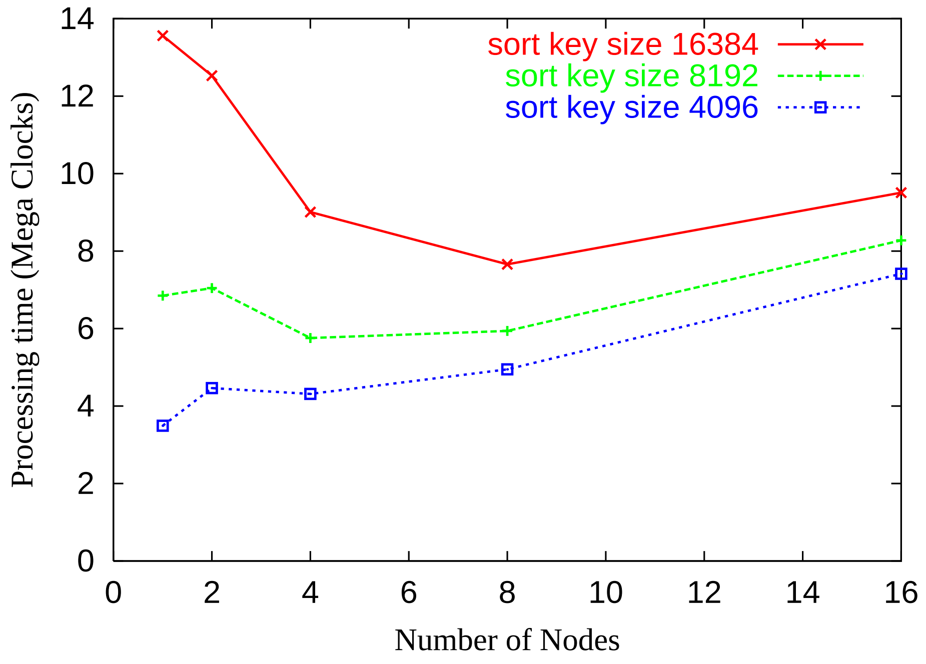

Figure 17: Processing Time of “radix sort” Program (Ring Length 100m).

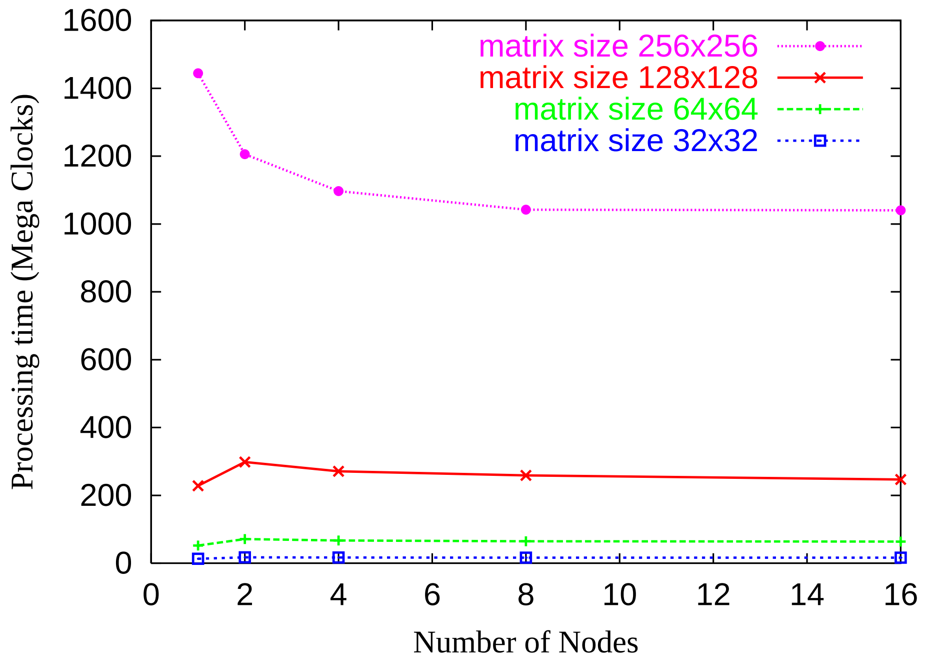

Figure 18: Processing Time of “product of a matrix” Program (Ring Length 1km).

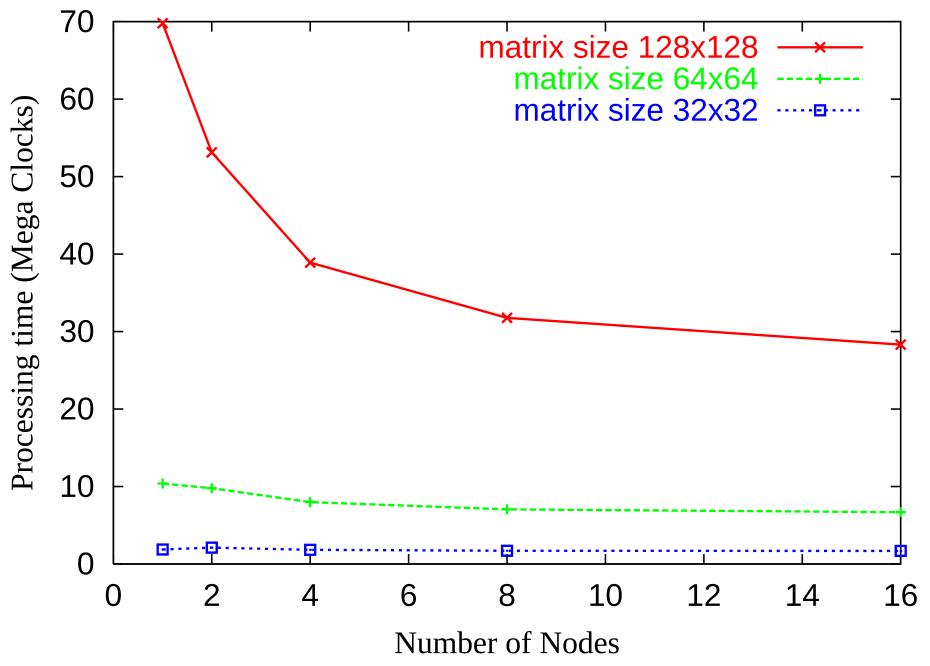

Figure 19: Processing Time of “product of a matrix” Program (Ring Length 100m).

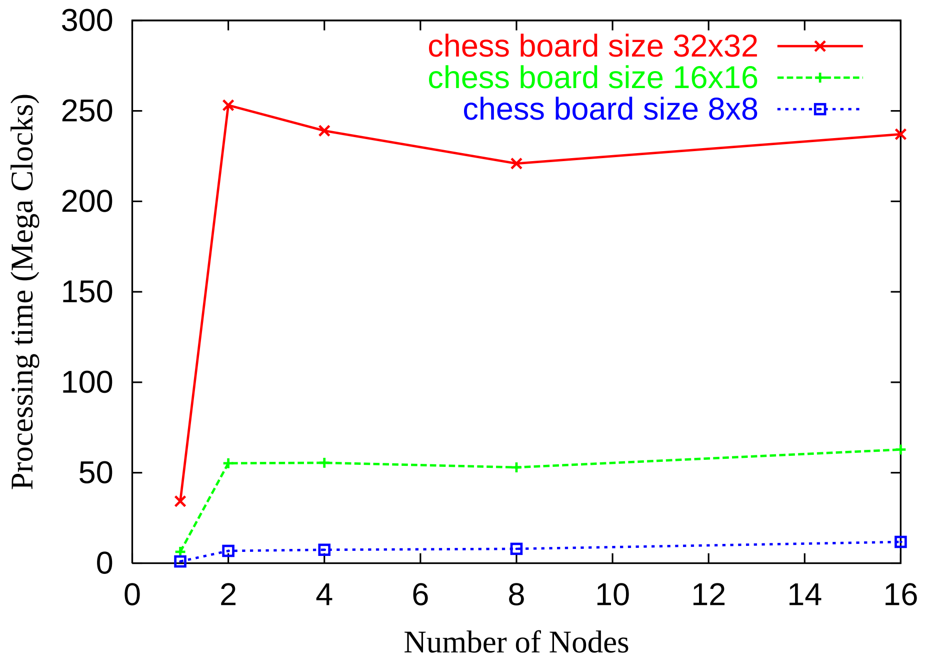

Figure 20: Processing Time of “queen problem” Program (Ring Length 1km).

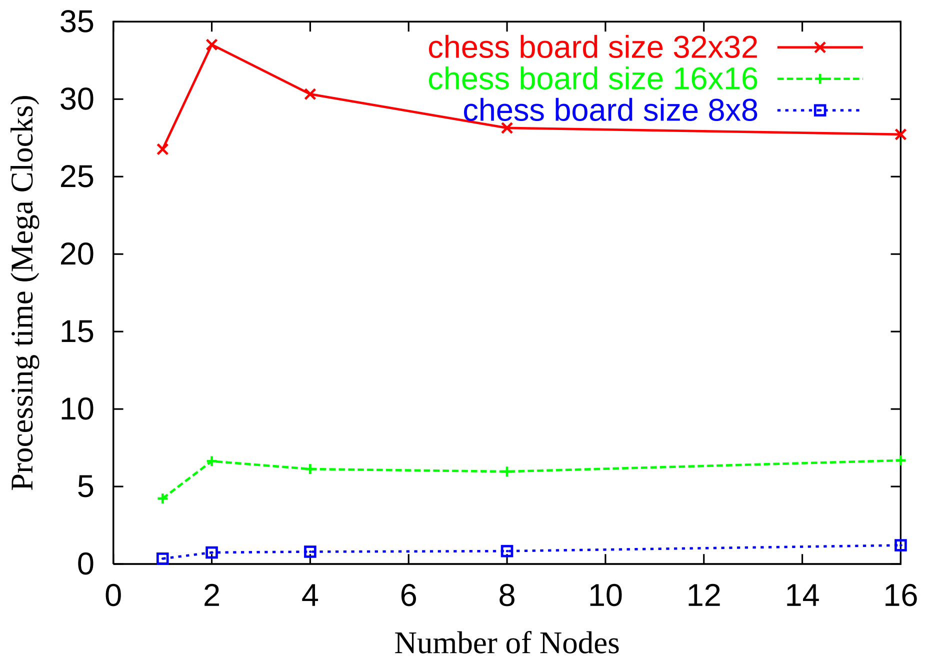

Figure 21: Processing Time of “queen problem” Program (Ring Length 100m).

4.4 Synchronization Improvement

As mentioned in Sec. 4.3, synchronization has a great influence on the performance of parallel computation. We thus propose a method to improve synchronization time. As mentioned in Sec. 3.4, we use barrier synchronization to enable collaboration between parallel nodes. The data used for barrier synchronization are stored on synchronous memory and the copy of data are stored on the local cache of the node. Since data used for barrier synchronization are referred by each node only once, the performance must be improved by not storing data on synchronous memory to the local cache. This is because cache coherency processing is not needed in this case, and it is sufficient for each node to read or write to synchronous memory only when data on synchronous memory are not stored in the local cache.

Lastly, we compare the performance of the above–mentioned caching policy to that of the original one. For this purpose, we present the speedup ratio of the new policy compared to the original one. The ring length is assumed to be 10km. The result shows that the speedup ratio of the “radix sort” program (Fig. 22) and the “product of a matrix” program (Fig. 23) are small. We show the speedup ratio for the “queen problem” program in Fig. 24. The larger speeding ratio can be detained by the increasing number of nodes. It is due to the fact that the “queen problem” program requires the largest number of accesses to the synchronous memory among the programs that we have tested. Then, the effect of the synchronization mechanism that we have introduced in this subsection becomes clear.

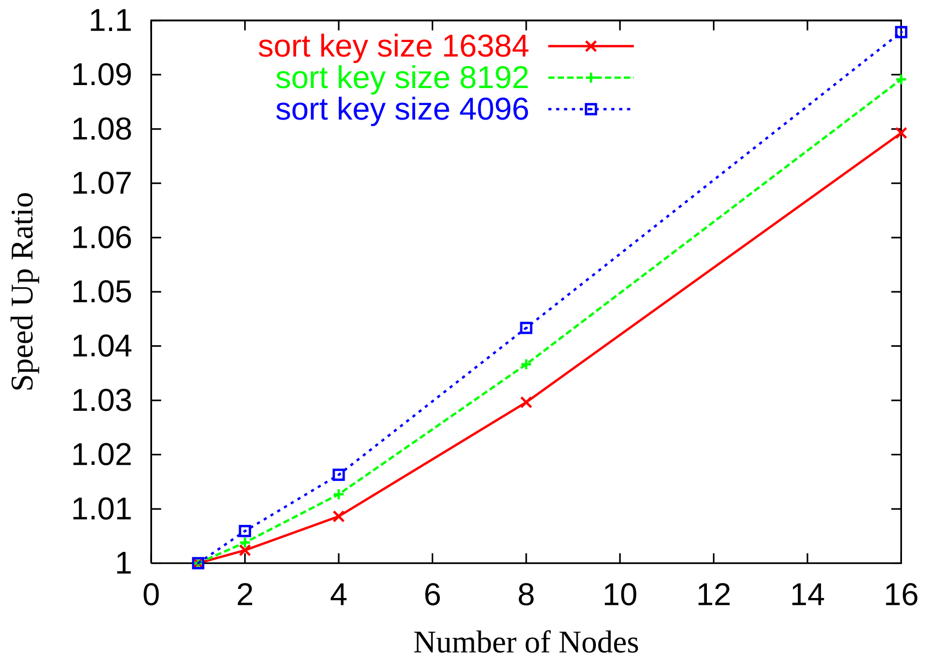

Figure 22: Speed Up Ratio of “radix sort” Program (Ring Length 10km).

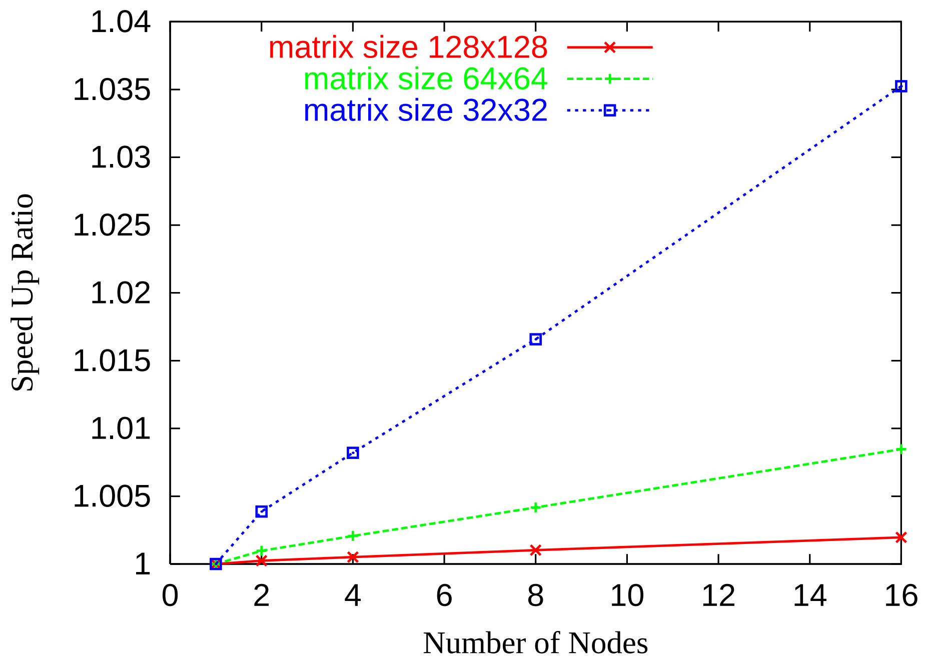

Figure 23: Speed Up Ratio of “product of a matrix” Program (Ring Length 10km).

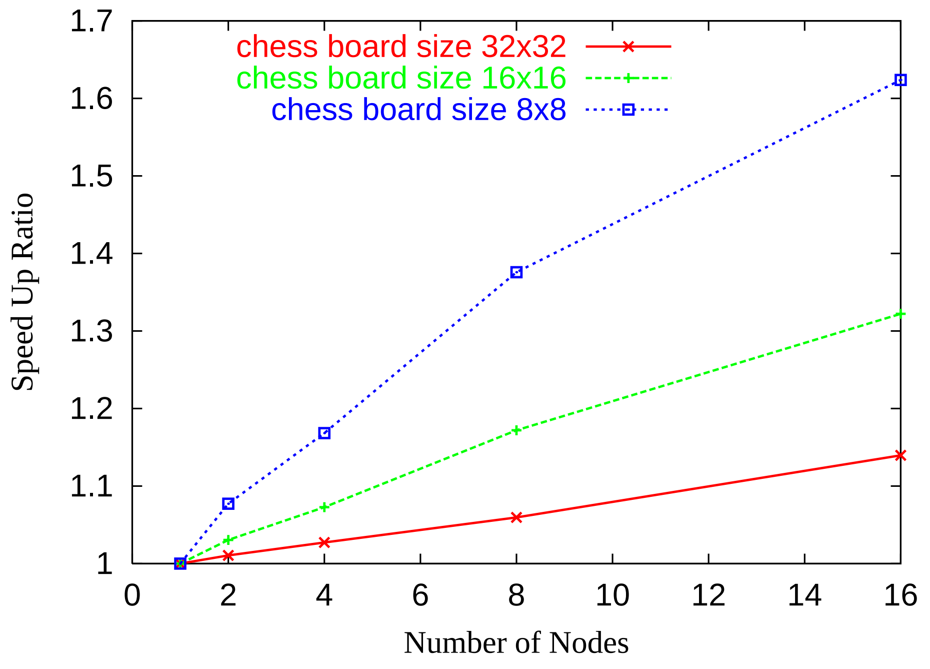

Figure 24: Speed Up Ratio of “queen problem” Program by Changing the Caching Policy of the Synchronous Memory (Ring Length 10km).

5 Improvement of Shared Memory Access Method

In this section, we explain the shared memory access methods when we use multiple control tokens to improve the performance of parallel computation. By using multiple control tokens, the access time to the control token is reduced. However, we use the control token to avoid the contention to the shared and synchronous memories as mentioned in Sec. 3.2. So when we use multiple control tokens, the avoidance of contention to the shared and synchronous memories becomes more complicated.

We propose two kinds of cache coherency protocols. In Sec. 5.1, we propose a cache coherency protocol that solves cache coherency and contention to the shared and synchronous memories by using multiple control tokens. In Sec. 5.2, we additionally introduce a cache coherency protocol where we use the lock bit to avoid the contention to the shared and synchronous memories.

5.1 Contention Avoidance Using Multiple Control Tokens

In this subsection, we explain the cache coherency protocol that solves cache coherency and contention to the shared and synchronous memories by introducing multiple control tokens. In this protocol, we use multiple control tokens to avoid the contention to the shared and synchronous memories as shown in Fig. 25. We attach the priority to each control token to decide which write request is given priority over other control tokens. The state transition of this protocol is the same as in Fig. 9. The processing of updating data on its local cache and writing data when a computing node does not have the copy cache block on its local cache (Fig. 9 (7), (9)) are different from the cache coherency protocol that uses one control token in Sec. 3.3. Hereafter, we explain these processing.

At first, if the state of cache block is CS (Fig. 9 (7)), contention avoidance processing is needed. After having the control token, it has to check whether a lock request message of another computing node is attached to the control token. If a lock request message is attached, it has to wait until an unlock request message is sent and then restart the write processing. If a lock message is not attached, it attaches a lock request message and an invalidation request message to the control token. And, it has to wait until the control token returns. During waiting the control token, if another control token comes, the computing node has to check whether a lock request

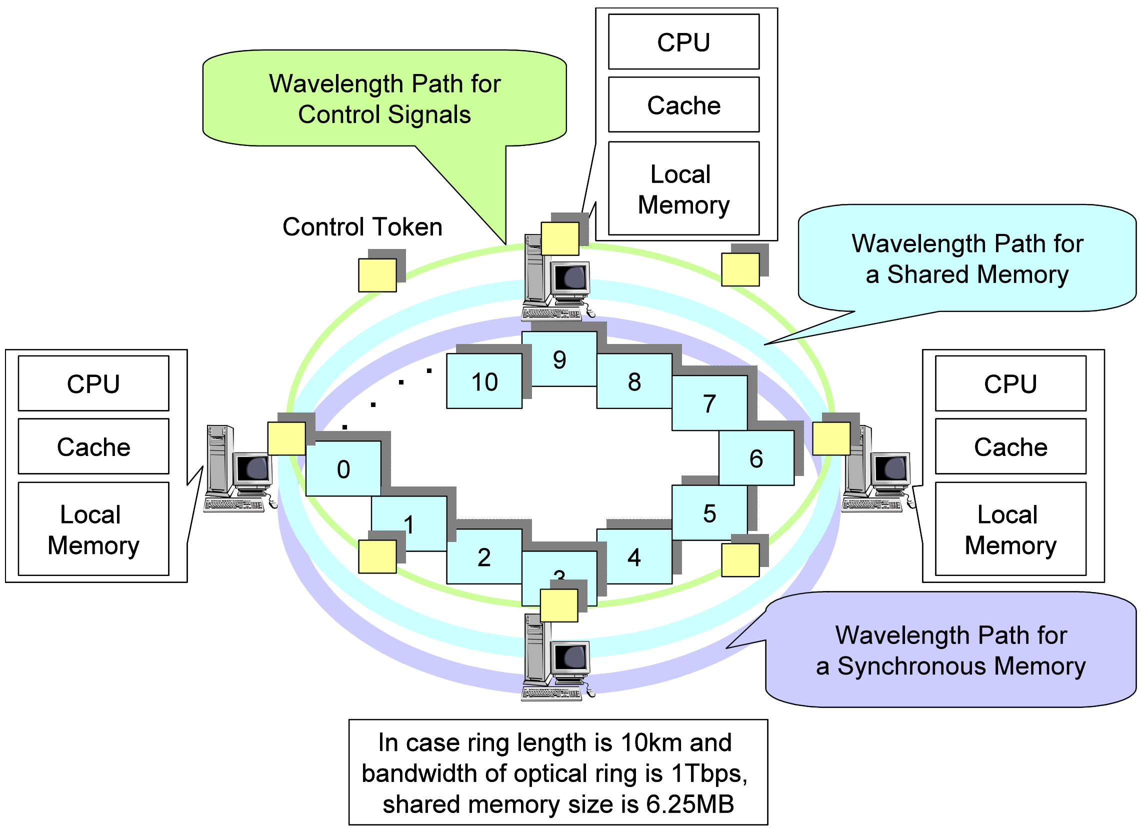
Wavelength Path for Control Signals
CPU
Cache
Local Memory
Control Token
Wavelength Path for a Shared Memory
CPU
Cache
Local Memory
CPU
Cache
Local Memory
10
9
8
7
6
5
4
3
2
1
0
Wavelength Path for a Synchronous Memory
In case ring length is 10km and bandwidth of optical ring is 1Tbps, shared memory size is 6.25MB

Figure 25: Network Model Using Multiple Control Tokens.

message of another computing node is attached to the control token. If a lock request message is attached and the priority of the control token is higher than the control token that the computing node attached a lock request message, it has to wait until an unlock request message is sent and then restart the write processing. Then it updates the data on the local cache and changes the state of the cache block ($CS \rightarrow D$). Finally, it attaches an unlock request message to the control token after having the control token.

Next, we explain how a computing node tries to write when it does not have the copy cache block on the local cache (Fig. 9 (9)). In this case, contention avoidance processing is also needed. A computing node searches the lock table. If the address of the cache block that it tries to write is registered in the lock table, it has to wait until an unlock request message is sent. If the address is not registered, it has to check whether a lock request message of other computing node is attached to the control token after having the control token. If a lock request message is attached, it has to wait until an unlock request message is sent and then restart the write processing. If a lock request message is not attached, it attaches a lock request message and a line move request message to the control token. It waits until the control token returns. During waiting the control token, if other control token comes, the computing node has to check whether a lock request message of another computing node is attached to the control token. If a lock request message is attached and the priority of the control token is higher than the control token that the computing node attached a lock request message, it has to wait until an unlock request message is sent and then restart the write processing. Then, if it receives a line copy ack, it has to wait until a cache block is sent by another computing node and read the cache block from the control token, and it updates the cache block and changes the state of the cache block ($I \rightarrow D$). If it does not receive a line copy ack, it directly reads the cache block from the shared memory. Then it updates the cache block and changes the state of the cache block ($I \rightarrow D$). After updating the cache block, it attaches an unlock request message to the control token after having the control token.

5.2 Contention Avoidance Using Lock Bit

In this subsection, we explain the cache coherency protocol that solves cache coherency and contention to the shared and synchronous memories by introducing multiple control tokens and a lock bit in stead of a lock request message. In this protocol, we use the lock bit to avoid the contention to the shared and synchronous memories as shown in Fig. 26. That is, we prepare the lock bit to each address of shared and synchronous memories. The state transition of this protocol is the same as in Fig. 9. The processing of updating the data on its local cache and writing the data when a computing node does not have the copy cache block on its local cache (Fig. 9 (7), (9)) are different from the cache coherency protocol that uses one control token in Sec. 3.3.

We describe how a computing node updates the data on a local cache. If the state of cache block is CS (Fig. 9 (7)), contention avoidance processing is needed. It has to wait until the relevant data on the shared memory comes in order to check whether the lock bit of the address is set by another computing node. If the lock bit is set, it has to wait until the lock bit is reset and then restart the write processing. If the lock bit is not set on the other hand, it sets the lock bit. After having the control token, it attaches an invalidation request message to the control token. Then it updates the data on the local cache and changes the state of the cache block ($CS \rightarrow D$). After updating the cache block, it has to wait until the relevant data on the shared memory comes and reset the lock bit.

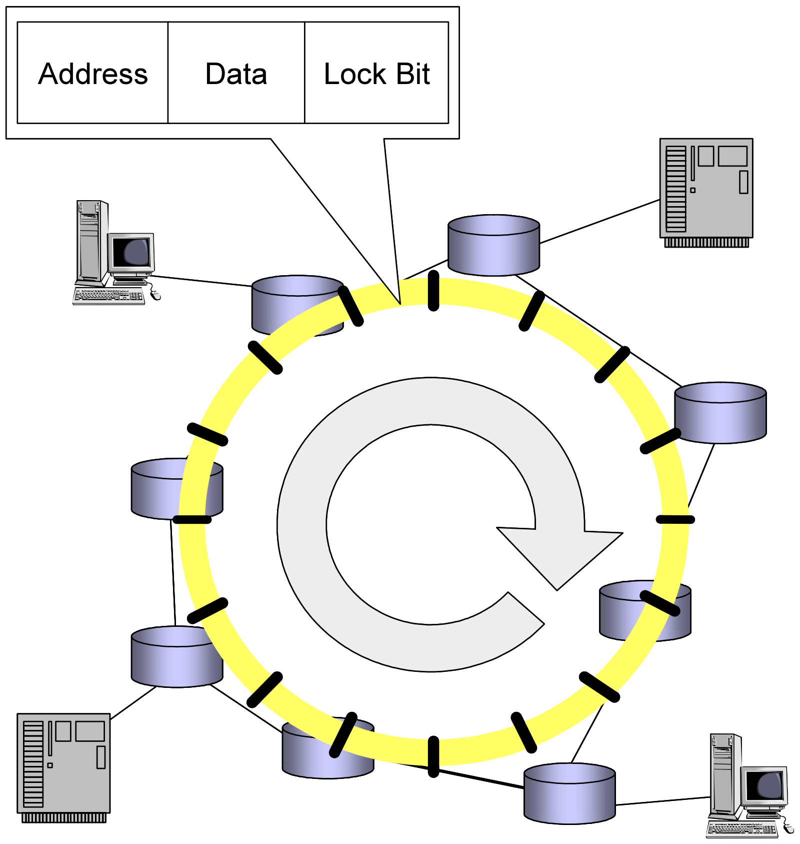

Figure 26: Cache Coherency Protocol Using a Lock Bit.

It is then illustrated how a computing node tries to write when it does not have the copy cache block on the local cache (Fig. 9 (9)). In this case, contention avoidance processing is also needed. It has to wait until the relevant data on the shared memory comes in order to check whether the lock bit of the address is set by another computing node. If the lock bit is set, it has to wait until the lock bit is reset and then restart the write processing. Otherwise, it sets the lock bit. After having the control token, it attaches a line move request message to the control token. Then it waits until the control token returns. If it receives a line copy ack, it has to wait until a cache block is sent by another computing node and read the cache block from the control token. Then it updates the cache block and changes the state of the cache block ($I \rightarrow D$). If it does not receive a line copy ack, it directly reads the cache block from the shared memory. Then it updates the cache block and changes the state of the cache block ($I \rightarrow D$). After updating the cache block, it has to wait until the relevant data on the shared memory comes and reset the lock bit.

Table 2: Access Delay Time of Shared Memory Access (Ring Length 10km).

	write hit (clocks)	write miss (clocks)	read hit (clocks)	read miss (clocks)
one control token	25,000	137,500	1	137,500
10 control tokens	52,500	103,750	1	103,750
lock bit	27,500	128,750	1	103,750

5.3 Estimation of the Execution Clocks

In Table 2, we show the access delay time to the shared memory when we use our proposal snoop cache protocols. In the cache coherency protocol using multiple control tokens, we use 10 control tokens. We also show level-1 cache hit ratio of each application program in Table 3. We estimate the execution clocks of shared memory access using the access delay time and level-1 cache hit ratio in Figs. 27, 28, and 29. In all application programs, the cache coherency protocol using a lock bit shows the best performance. On the other hand, the cache coherency protocol using multiple control tokens shows the worse performance compared to other protocols. This is because the cache coherency protocol using multiple control tokens takes longer time to avoid the contention to memories than the cache coherency protocol using one control token and a lock bit as shown in Table 2. Making a simulator of using a lock bit and evaluating the performance through simulation is left to be in the future task.

Table 3: Characteristics of Hit Ratio of Shared Memory Access (Ring Length 10km).

	nodes	shared write accesses	shared read accesses	shared write hit ratio	shared read hit ratio
radix sort (16384 keys)	1	50,176	66,048	83.54%	100.00%
	2	50,726	272,296	78.18%	85.47%
	4	51,766	773,530	75.18%	89.28%
	8	53,846	2,232,960	72.14%	93.66%
	16	58,006	8,154,045	65.81%	96.85%
product of a matrix (128×128)	1	49,152	32,768	75.00%	100.00%
	2	49,162	16,429,802	75.00%	99.89%
	4	49,162	47,224,401	75.00%	99.92%
	8	49,162	108,824,063	75.00%	99.93%
	16	49,162	231,989,561	75.00%	99.94%
queen problem (32×32)	1	141,675	357,575	99.60%	100.00%
	2	142,290	1,254,355	97.91%	64.06%
	4	142,692	1,614,602	94.45%	63.94%
	8	143,496	3,552,489	94.41%	52.20%
	16	145,104	7,829,081	94.32%	67.80%

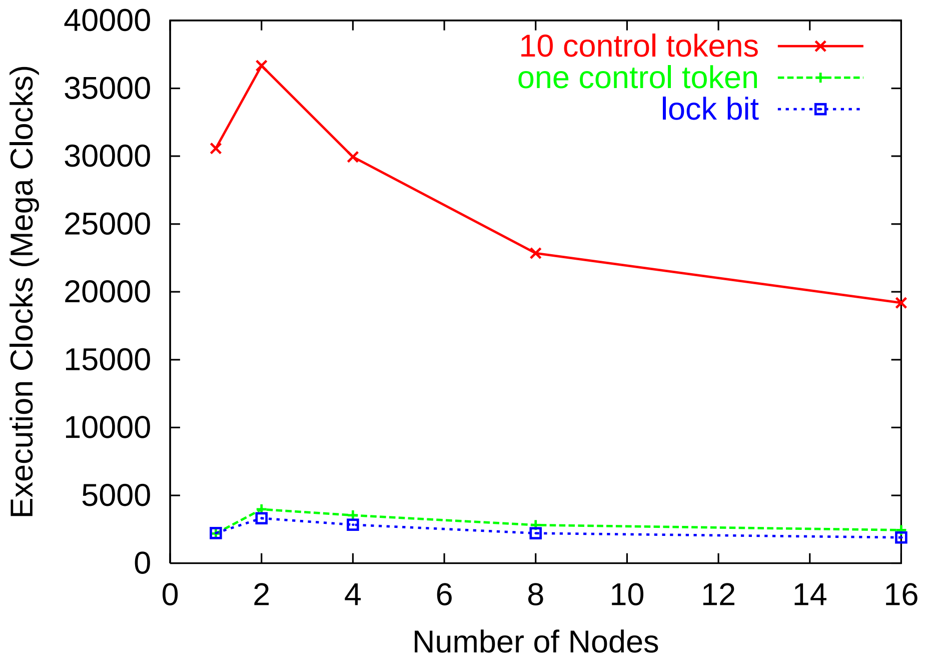

Figure 27: Execution Clocks of Shared Memory Access (“radix sort” Program).

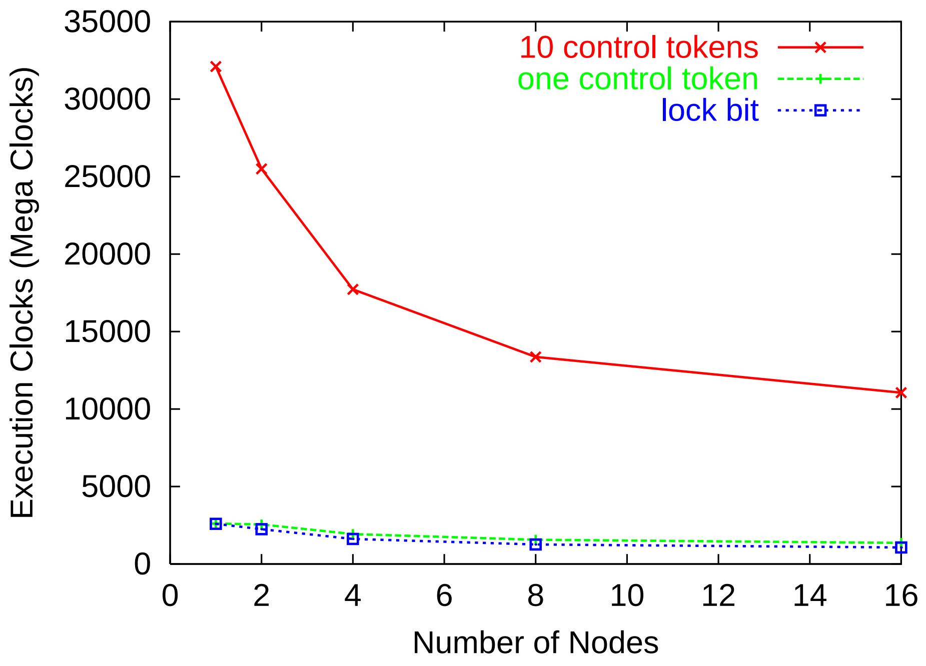

Figure 28: Execution Clocks of Shared Memory Access (“product of matrix” Program).

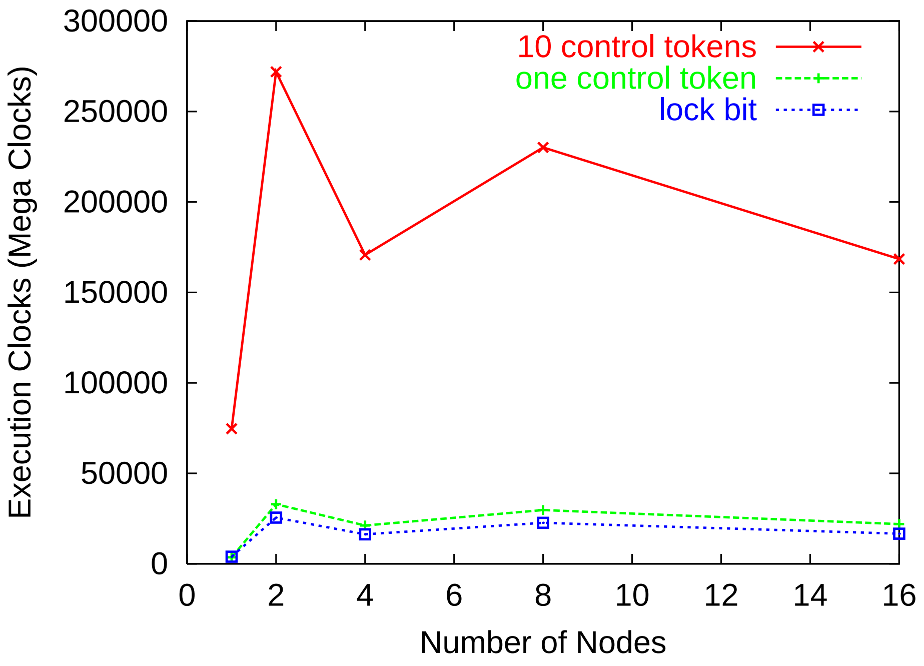

Figure 29: Execution Clocks of Shared Memory Access ("queen problem" Program).

6 Conclusion

In this thesis, we proposed a new computing environment (which we will refer to λ computing environment) that provides a base for parallel computing among nodes distributed in the wide area. We have proposed the shared memory access method in realizing the shared memory on photonic network. Moreover, we have evaluated the performance of the proposed method using the benchmark program for parallel computing. As a result, we show that the effectiveness of using optical ring as a shared memory and of parallel processing by the increase in the number of nodes when number of synchronous processing is small. We can see the future possibility of all-optical parallel computing environment in the wide-area. An efficient shared memory access method and a practical use of a local memory is due to be considered in the future. Furthermore, in this simulation, since neither the processing delay by the interface nor simultaneous access to a photonic ring network from two or more node computers is taken into consideration, it is the subject which should be improved.

Acknowledgements

I would like to express my deepest gratitude to my supervisor Professor Masayuki Murata at Osaka University, who introduced me to the area of computer networks including the subjects in this thesis and support through my studies. His excellent guidance and thoughtful advice have provided an invaluable experience that will help me in my career.

All works of this thesis would not have been possible without the helpful suggestions and support of Associate Professor Ken-ichi Baba at Osaka University. He has opened the way for the accomplishment of this thesis.

I am also indebted to Associate Professor Naoki Wakamiya, Associate Professor Go Hasegawa, Assistant Professor Ichinoshin Maki and Assistant Professor Shin'ichi Arakawa at Osaka University who gave me helpful comments and feedbacks. Furthermore, I thank all my colleagues in the Advanced Network Architecture Research Group for their generous help, enlightening and valuable suggestions.

For simulator program coding, we received much useful advices from Mr. Masaki Wakabayashi (present by Sony Corporation) of the Keio University, Science and Technology graduate course. The authors wish to thank Mr. Masaki Wakabayashi.

And this work was supported in part by the National Institute of Information and Communications Technology (NiCT) in Japan, and by a Grant–in–Aid for Scientific Research from the Miqnistry of Education, Culture, Sports, Science and Technology of Japan.

Finally, I wish to express my warmest thanks to my parents, grandparents and sister for their unconditional love, support, patience and understanding.

References

[1] M. Murata and K. Kitayama, “A 1,000-channel WDM network can resolve network bottleneck,” in *Proceedings of the 7th Asia-Pacific Conference on Communications (APCC 2001) (Tokyo)*, pp. 113–116, Sept. 2001.

[2] E. L. Berger, “Generalized multi-protocol label switching (GMPLS) signaling functional description,” *IETF RFC3471*, Jan. 2003.

[3] T. Yamaguchi, K. Baba, M. Murata, and K. Kitayama, “Packet scheduling for WDM fiber delay line buffers in photonic packet switches,” in *Proceedings of OptiComm 2002*, vol. 4874, pp. 262–273, July 2002.

[4] K. Baba, R. Takemori, M. Murata, and K. Kitayama, “A packet scheduling algorithm for the 2x2 photonic packet switch with FDL buffers,” in *Proceedings of 28th Europian Conference on Optical Communication 2002 (ECOC2002)*, Sept. 2002.

[5] S. L. Danielsen, B. Mikkelsen, C. Joergesen, T. Durhuus, and K. E. Stubkjaer, “WDM packet switch architectures and analysis of the influence of tuneable wavelength converters on the performance,” *IEEE Jounal of Lightwave Technology*, vol. 15, pp. 219–227, Feb. 1997.

[6] S. L. Danielsen, C. Joergesen, B. Mikkelsen, and K. E. Stubkjaer,, “Analysis of a WDM packet switch with improved performance under bursty traffic conditions due to tuneable wavelength converters,” *IEEE Journal of Lightwave Technology*, vol. 16, pp. 729–735, May 1998.

[7] D. Hunter, M. C. Chia, and I. Andonovic, “Buffering in optical packet switches,” *IEEE Journal of Lightwave Technology*, vol. 16, pp. 2081–2094, Dec. 1998.

[8] K. L. Hall and K. A. Rauschenbach, “All-optical buffering of 40-gb/s data packets,” in *IEEE Photonic Technology Letters*, vol. 10, pp. 442–444, 1998.

[9] T. Yamaguchi, K. Baba, M. Murata, and K. Kitayama, “Scheduling algorithm with consideration to void space reduction in photonic packet switch,” *IEICE Transactions on Communications*, vol. E86-B, pp. 2310–2318, Aug. 2003.

[10] T. DeFanti, M. Brown, J. Leigh, O. Yu, E. He, J. Mambretti, D. Lillethun, and J. Weinberger, "Optical switching middleware for the OptIPuter," *IEICE Transaction on Communication*, vol. E86-B, Aug. 2003.

[11] H. Nakamoto, K. Baba, and M. Murata, "Shared memory access method for a λ computing environment," in *Proceedings of IFIP Optical Networks and Technologies Conference (OpNeTec)*, pp. 210–217, Oct. 2004.

[12] E. Taniguchi, K. Baba, and M. Murata, "Implementation and evaluation of shared memory system for establishing lambda computing environment," Tech. Rep. 255, IEICE, Aug. 2004.

[13] H. Amano, *Parallel Computer*. Shoukoudou, June 1996.

[14] N. Suzuki, S. Shimizu, and N. Yamanouchi, *An Implemantation of a Shared Memory Multiprocessor*. Koronasha, Mar. 1993.

[15] M. S. Papamarcos and J. H. Patel, "A low-overhead coherence solution for multiprocessors with private cache memories," in *Proceedings of IEEE 11th Annual International Symposium on Computer Architecture*, (New York), pp. 348–354, 1984.

[16] S. S. Thakkar and M. Sweiger, "Performace of an OLTP application on symmetry multiprocessor system," *17th Annual International Symposium on Computer Architecture*, pp. 228–238, May 1990.

[17] M. Wakabayashi, K. Inoue, and H. Amano, "ISIS: Multiprocessor simulator library," *Applied Informatics AI'99*, pp. 198–200, Feb. 1999.

[18] M. Wakabayashi and H. Amano, "Environment for multiprocessor simulator development," *I-SPAN 2000*, pp. 64–71, Dec. 2000.

[19] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 programs: Characterization and methodological considerations," in *Proceedings of the 22nd International Symposium on Computer Architecture*, pp. 24–36, June 1995.