**Master's Thesis**

Title

# Design and Evaluation of Shared Memory Architecture for WDM-based $\lambda$ Computing Environment


Supervisor

Professor Masayuki Murata

Author

Eiji Taniguchi


February 15th, 2006


Department of Information Networking

Graduate School of Information Science and Technology

Osaka University

Master's Thesis

**Design and Evaluation of Shared Memory Architecture**
**for WDM-based $\lambda$ Computing Environment**

Eiji Taniguchi

## Abstract

Grid computing, where we can compute large scale problems which we cannot solve with only single computer, has been studied and developed actively. We usually treat volume data in Grid computing environment, so we need to transfer such data in high speed and with high reliability. In conventional TCP/IP in the Internet, it is difficult to achieve good performance because of overhead caused by packet processing and retransmission of lost packets.

So, we have proposed $\lambda$ computing environment. In $\lambda$ computing environment, network switches and computing nodes are connected each other with optical fibers, and by establishing optical wavelength paths between end hosts, we can offer high speed and high reliable communication pipe for data sharing or data exchanging between computing nodes. Here we need to consider shared memory architecture to solve large scale problems utilizing communication pipe in $\lambda$ computing environment. That is, it is different from the architecture of the conventional multi-processor system or cluster system because computing nodes are located in a wide area in $\lambda$ computing environment. So that the performance of networks affects the performance of shared memory and computing power.

In this study, we model and analyze the shared memory architecture in $\lambda$ computing environment, and we show how the network topology and the control method for cache coherency influence the performance. Here we use semi-Markov process which enable us to set state residence time for modeling, and evaluate what type of shared memory architecture is suitable for $\lambda$ computing environment. As a result, we found that we could achieve high performance with some parameter regions or conditions in each shared memory architecture.

**Keywords**

$\lambda$ computing environment

Shared memory architecture

Cache coherency

Semi–Markov process

## Contents

## List of Figures

## List of Tables

# 1 Introduction

Recently, the demand for large scale computation such as the gene information analysis, image processing and the global environment simulation that treats the volume data is arising. In order to execute large scale computation, the Grid computing technology has been actively studied and developed. We expect to make distributed parallel processing and to calculate effectively by utilizing CPUs and storage of computing nodes connected by networks in Grid computing environment. In Grid computing environment, TCP/IP is usually used for communication such as control messages and data exchanges between computing nodes. However TCP/IP has some harmful effects in such environment. For example, some packets may be lost on the route from the source node to the destination node because of traffic congestion caused by own volume data transmission on the high-speed network. So that it needs retransmission of lost packets and then causes degradation of network throughput and computing throughput on Grid.

To satisfy the demand in Grid computing, the new technology that enables high-speed and high reliable communication is needed, so that research in optical domain has been studied in recent years. Especially, the WDM (Wavelength Division Multiplexing) technology that use multiplexed light wavelengths in optical domain is focused on. And 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 (Generalized Multi-Protocol Label Switching), which is the communication technology that uses various optical technologies for a lower layer than the WDM technology, has also been advanced in IETF [1].

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. In order to execute distributed computing effectively with Grid technologies which realize a volume data transfer on the photonic network, new architecture which is different from conventional architecture is required.

Therefore, we consider that we establish broadband wavelength paths between computing nodes and then provide these paths for end users as a realization method to achieve high-speed and high reliable communication in Grid computing environment. That is, it is possible to provide

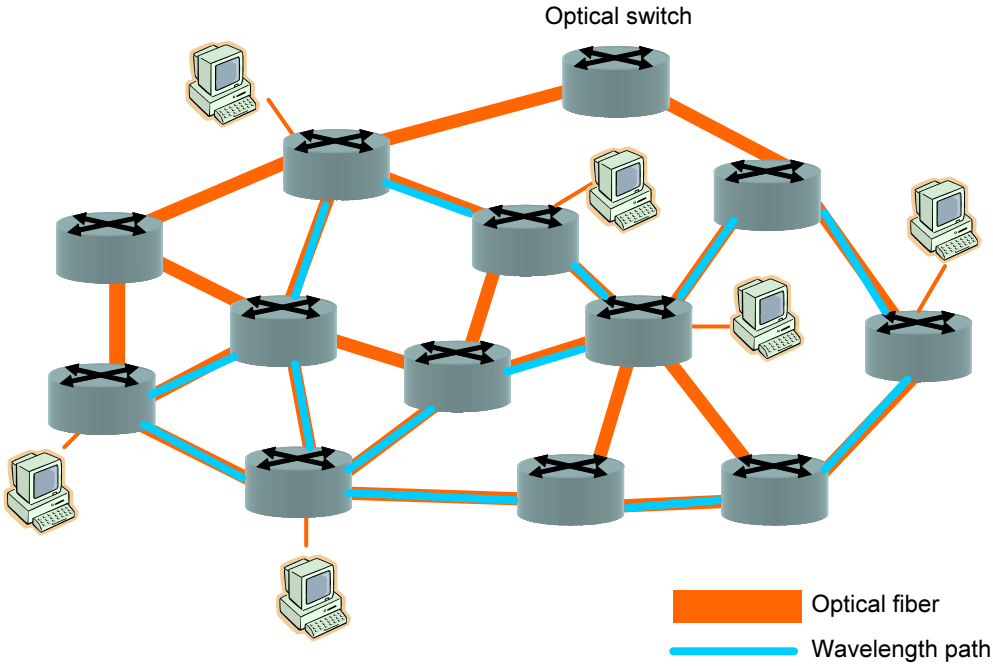



Figure 1: Brief overview of $\lambda$ computing environment.

an end user with ultra high speed and high reliable communication pipe by building a photonic network that uses established fibers, or newly laid fiber if needed, and by utilizing wavelength paths multiplexed in the fibers as the minimum particle size for information exchanges. Thus we propose a new distributed computing architecture which we call $\lambda$ computing environment. In $\lambda$ computing environment, by connecting computing nodes and optical switches on the photonic network with the optical fibers each other (See Fig. 1), we can provide exclusive wavelength paths and then computing nodes perform distributed computing by using these paths as exclusive communication pipe. In $\lambda$ computing environment, we can realize high-speed and high reliable data exchanging or data sharing because computing nodes utilize not conventional TCP/IP network but utilize beforehand established wavelength paths as exclusive communication channel.

Related works [2, 3] report which evaluate the architecture that realize distributed parallel computing in $\lambda$ computing environment. Both presume shared memory architecture for data sharing that is required for parallel computing. In [2], they make the virtual optical ring network connected wavelength paths between computing nodes, and utilize the ring network in itself as a shared memory. Shared memory architecture introduced in [3] are more realistic architecture than

that of [2]. In [3], every computing node has own shared memory and the data on the shared memory is the same in shared memory over all computing nodes. This shared memory is connected to photonic network, and updated data on the shared memory is reflected to the shared memory on other computing nodes through photonic network.

However, these studies does not discussed the influence of network characteristics or cache coherency protocols to the performance of the architecture though these evaluated shared memory architecture based on execution time of parallel application programs through simulation or as an example of implementation on actual computers. In $\lambda$ computing environment, an architecture which assumed to have a shared memory makes it easy for us to make a programming coding of parallel computing applications, however it needs longer time for data sharing than a conventional multi processor computer or a cluster computer with SDSM (Software Distributed Shared Memory) because computing nodes are located in a wide area. Therefore, data sharing processing may influence to the computation performance. So, we have to consider data sharing methods, such as access methods to a shared memory and cache coherency protocols more than conventional multi processor systems. Moreover, we have to understand how the characteristics of the network, network topologies and cache protocols have an impact on the shared memory architecture.

In this study, we design some types of shard memory architecture in terms of network topology, memory access model and cache protocol in $\lambda$ computing environment, and we model and analyze these architecture by using semi-Markov process which enable us to set state residence time for modeling. And through numerical example, we clarify how the network topology and the control method for cache coherency influence the performance and evaluate what type of shared memory architecture is suitable for $\lambda$ computing environment.

The rest of the thesis organized as follows. In Section 2, we explain $\lambda$ computing environment that we proposed. In Section 3 we design the shared memory architecture for $\lambda$ computing environment and in Section 4 we model and analyze the model with semi-Markov process. Then we show the evaluation through numerical examples in Section 5 and we conclude this study in Section 6.

## 2 $\lambda$ computing environment: A new distributed computing environment

As described in section 1, the performance of shared memory architecture may be influenced by network topologies, memory access models, cache coherency protocols and so on in $\lambda$ computing environment where computing nodes are located in a wide area not like a conventional multi processor system. Therefore, we cannot sweepingly decide what types of architecture is suitable for $\lambda$ computing environment.

In this section, in order to investigate which architecture is suitable for $\lambda$ computing environment, we firstly explain about $\lambda$ computing environment that we have proposed as new distributed computing environment and about the reason why we presume shared memory architecture. Secondly, we explain about overview of our proposed shared memory architectures. Then, we denote that considering factors such as network topologies, memory access models, cache coherency protocols exist in $\lambda$ computing environment and how they can influence on the performance of shared memory architecture.

### 2.1 Basic technology

$\lambda$ computing environment is based on WDM technology. Computing nodes and optical switches that compose $\lambda$ computing environment are connected with optical fibers. In a optical fiber, 100 or more wavelengths, 1000 or more in a future, are multiplexed by WDM or DWDM (Dense WDM) technology and provide broadband communication line for computing nodes. WDM technology is usually considered as a lower layer technology that realize GMPLS and IP over WDM network. In this study, we use WDM technology for establishing wavelength paths and utilize their paths as exclusive communication line.

Therefore in $\lambda$ computing environment, we can realize high-speed and high reliable data exchanging or data sharing because computing nodes utilize not conventional TCP/IP network but utilize beforehand established wavelength paths as exclusive communication channel. We show the detail of established wavelength paths in Figure 2.

### 2.2 Requirement to realize distributed computing environment

Next, we explain about how distributed computing is realized in $\lambda$ computing environment.

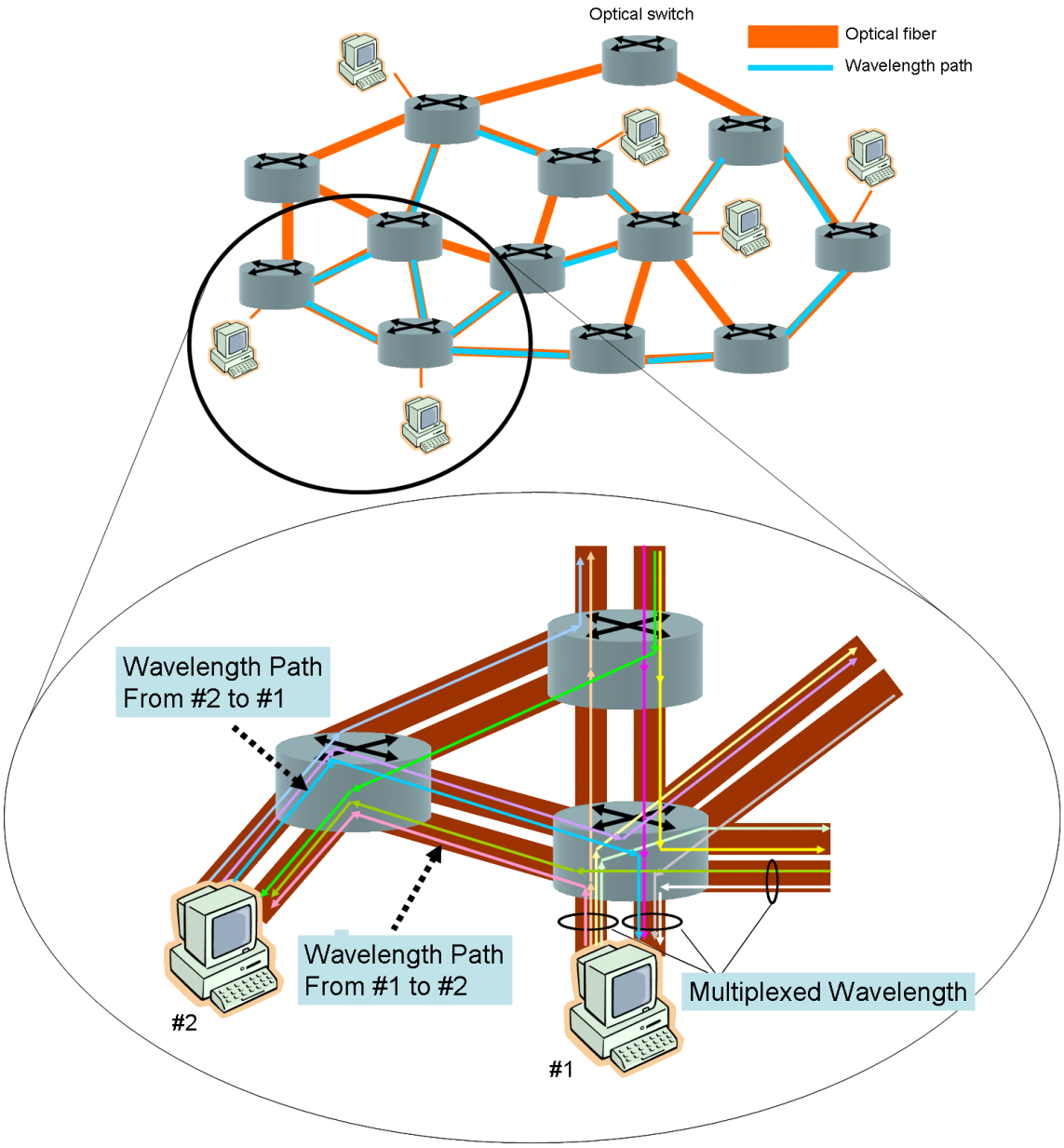



Figure 2: Established wavelength paths.

### 2.2.1 Data sharing method

When we try to realize and perform distributed computing in $\lambda$ computing environment, we have to consider that how data sharing among computing nodes is realized. We suppose two ways to realize data sharing. One is shared memory and another is distributed memory.

In distributed memory architecture, MPI (Message Passing Interface) [4] is usually used for realizing data sharing and data exchanging in parallel application. One representative implementation is MPICH which is often used over TCP/IP network. When we want to share a data in a MPI program, we usually use two typical function; MPI_Send and MPI_Recv like this.

```
shared int x;
    if (source_node) then
        MPI_Send(&x, 1, MPI_INT, destination_node, MPI_ANY_TAG,
                MPI_COMM_WORLD);
    else /* if destination_node */
        MPI_Recv(&x, 1, MPI_INT, source_node, MPI_ANY_TAG,
                MPI_COMM_WORLD, &stat);
```

In this code, in order to realize data sharing, source node send the shared variable $x$ by explicitly calling function, MPI_Send, and destination node receives the data and stores it to the shared variable $x$ by MPI_Recv.

On the other hand, in a shared memory architecture, we have only to use substitute expression like this

```
shared int x;
    x = 100;
```

Only executing this code, data sharing is implicitly done over all computing nodes. If other computing nodes want to use this value, they only have to read the data from $x$.

In this study, we focus on shared memory architecture. However, data sharing realized by message passing is also important because there is a lot of parallel application which use MPI. By realizing message passing architecture in $\lambda$ computing environment, we can reuse MPI programs without code modification and execute them in photonic network. One example of message passing architecture in $\lambda$ computing environment is introduced, implemented and evaluated in [5].

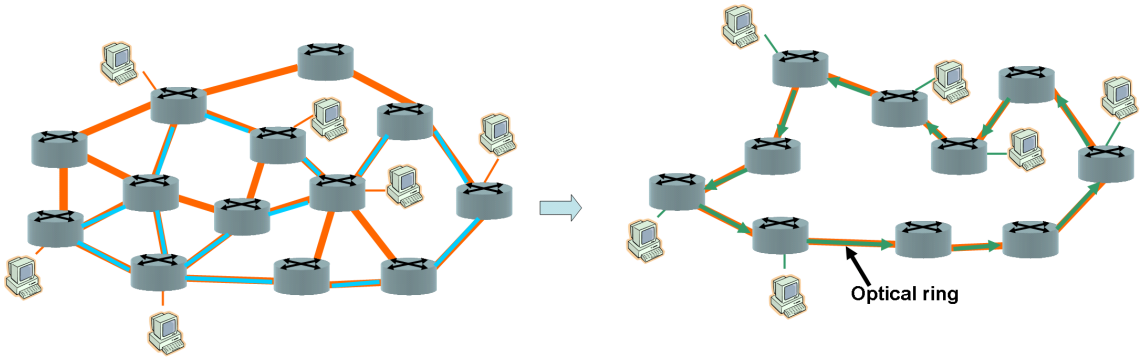



Figure 3: Virtual ring topology.

### 2.2.2 Data transmission method

Next, we explain about how the data updated at one node is reflected to other computing node. As described above, we utilize virtual channels for distributed computing. When we update data, the updated data is put onto virtual channels and send to other computing node. This proceeds implicitly in background work whenever we update shared data.

In the case that the virtual channels can constitute a virtual ring topology (See Fig. 3), the updated data go round on a ring, and all computing node can receive this data. So, we can realize updates among computing nodes by broadcasting data on the ring in $\lambda$ computing environment. On the other hand, the virtual channels can also constitute mesh topology which can be dynamically changed by selecting appropriate wavelengths multiplexed in optical fibers. Then, the virtual channels can organized optimal topology for data transmission and multicast when data sharing is needed (See Fig. 4). In this case, each computing node can directly communicate with each other and share the updated data.

## 2.3 Characteristic factor of shared memory architecture

As described in section 2.2.1, we focus on a shared memory architecture. So, we introduced what characteristic factors exist in shared memory architecture and how they can affect on the shared memory architecture. Here we focus on four characteristic factors, that is, network topology, memory access model, cache coherency protocol, and realization method for cache coherency protocol.

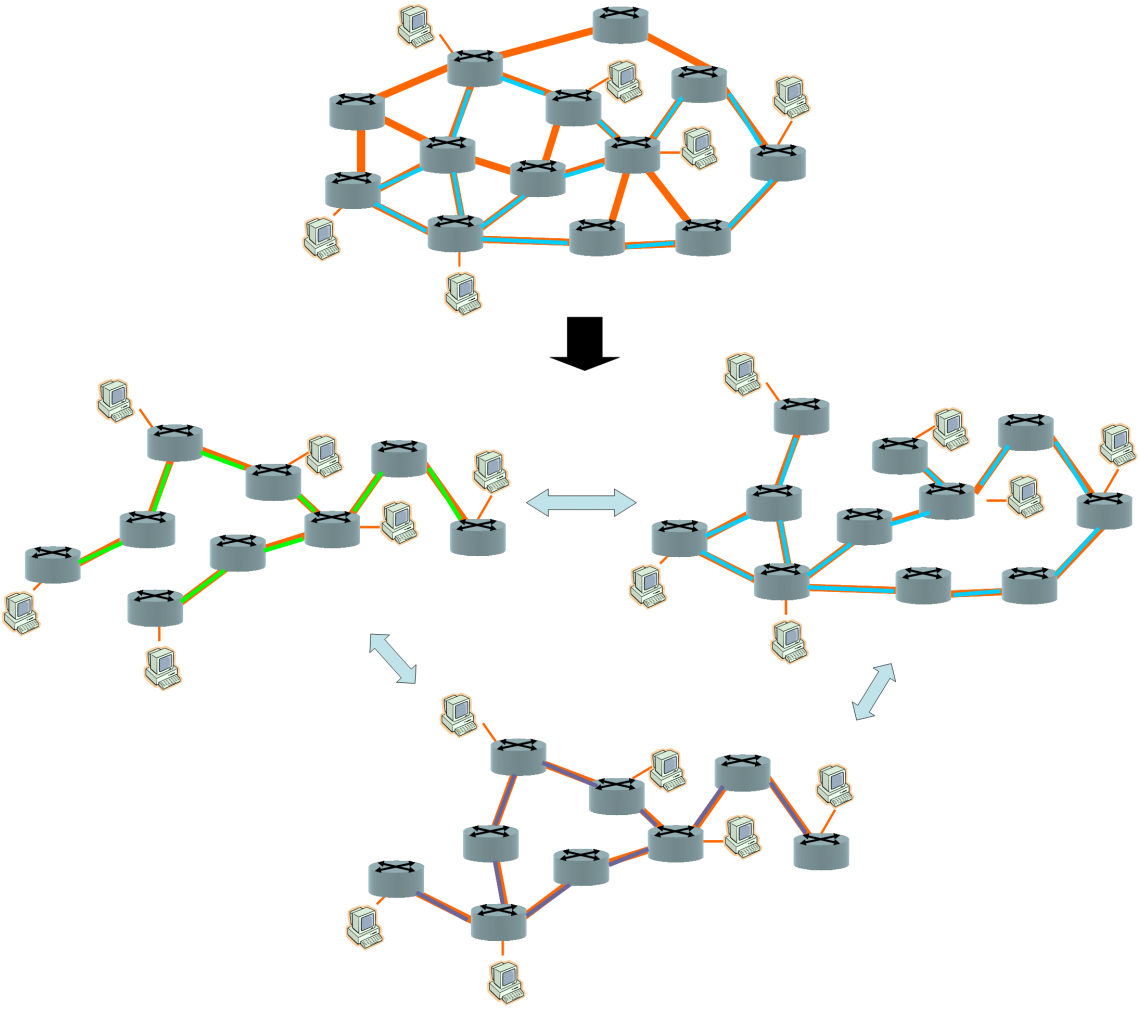


Figure 4: Mesh topology can be dynamically changed.

### 2.3.1 Network topology

A network topology can affect on propagation delay between computing nodes in $\lambda$ computing environment. In addition, this is also consideration matter for us to decide paths for data transmission and required number of wavelength. We introduce representative two topologies below.

- Ring topology

  This topology makes it easy to broadcast data for computing nodes. However, broadcasting data requires at least the time to round the ring once. Moreover, each computing nodes only forwards the data from upstream computing node to downstream computing node without data duplicating.

- Mesh topology

  Generally, average propagation delay on this topology become shorter than that of ring topology. However, most of computing nodes have to duplicate received data in order to forward their neighbor because they have more than two computing nodes at downstream.

### 2.3.2 Memory access model

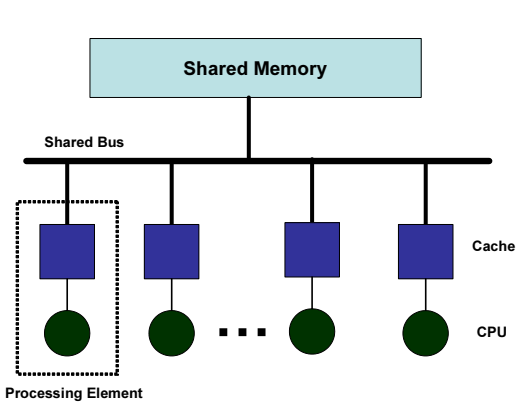
Memory access model provides how computing nodes access to the shared memory. That is, memory access model decides whether computing nodes can access to the shared memory directly or not. In the case that a computing node accesses to the shared memory through network, it takes longer time than the case that a computing node access to the shared memory without using network. Memory access model is roughly classified into three models. We describe about these models below.

- UMA (Uniform Memory Access) model

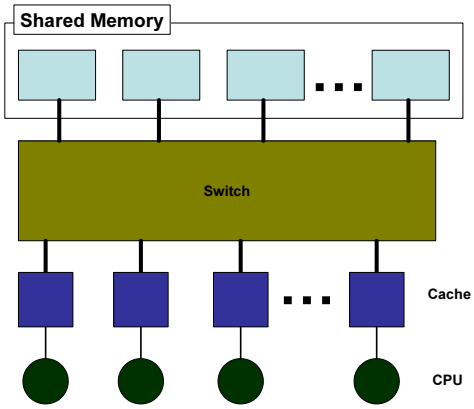
  In this model, all processing elements share entire address space, and access time to the shared memory is same (Fig. 5(a), Fig. 5(b)) .

- NUMA (Non Uniform Memory Access) model

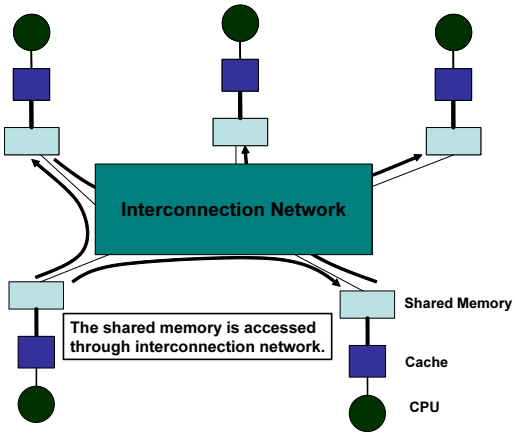
  In this model, all processing elements share entire address space, but access time to the shared memory depends on address where a processing element access to (Fig. 5(c)).



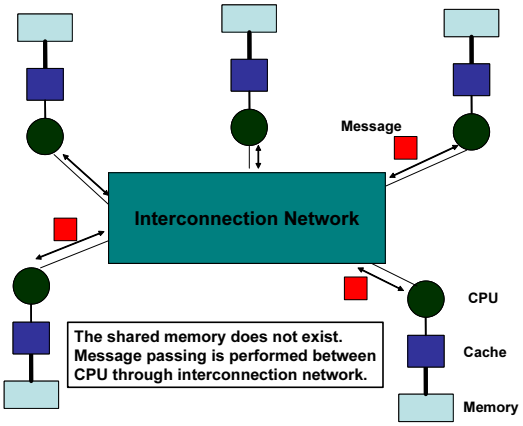

(a) UMA with shared bus




(b) UMA with switch




(c) NUMA




(d) NORMA

Figure 5: Memory access models.

- NORMA (NO Remote Memory Access) model

  In this model, each processing element has own memory which address space is independent each other. That is, a shared memory does not exist in this model and each processing element perform parallel computation by message passing (Fig. 5(d)).

When we adopt UMA model to $\lambda$ computing environment, read access does not need to use network, but write access needs to use network in order to update data on other shared memory. In NUMA model, in addition to write access, read access can also need to use network for accessing data on other shared memory. NORMA model, include message passing model, is out of scope

of this study because there is no shared memory in this model, and our target is a shared memory architecture.

### 2.3.3 Cache coherency protocol

When a processor tries to read or write to a shared memory, coherency controls 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 to keep cache coherency. Moreover, when a processor writes or updates the data on its cache, keeping cache coherency becomes still more complicated and there are some ways to keep coherency. Such cache coherency protocols are classified into four types according to the timing (write-through, and write-back) and the method (invalidation, and updating) [6].

- Write-through invalidation protocol

  Whenever a CPU write to the cache, the relative data on its shared memory is updated and the relative cache line that is kept by other cache memory is invalidated. It is easy to maintain cache coherency in this protocol. However, the performance is generally not better than that of write-back protocol because shared media such as shared bus is used whenever writing occurs, and congestion of shared media increases.

- Write-through updating protocol

  Whenever a CPU write to the cache, the relative data on its shared memory is updated and the relative cache line that is kept by other cache memory is also updated, as a result, cache coherency is achieved. However, this protocol belongs to write-through protocol type, so it has same problems mentioned above.

- Write-back invalidation protocol

  Whenever writing occurs, the relative cache line kept by other cache memory is invalidated, however in this protocol, the relative data on its shared memory is not updated. Therefore, at most only one cache memory which is updated by its CPU keeps the latest data. Thus, cache coherency is kept. This protocol is adopted many systems because processing and implementation of this protocol is easy.

- Write-back updating protocol

Whenever writing occurs, the relative cache line kept by other cache memory is updated. On the other hand, the relative data on its shared memory is not updated in this protocol.

When we adopt these protocols described above to $\lambda$ computing environment, the write-back invalidation protocol is most suitable because this protocol has the least network utilization. When we adopt the write-through cache to $\lambda$ computing environment, we have to use network in order to update data on other shared memory whenever writing access occurs. Moreover, when we adopt the write-back updating protocol to $\lambda$ computing environment, we also have to use network in order to update data on other cache memory whenever writing access occurs. On the other hand, in the write-back invalidation protocol, we do not use network when writing access occurs on a cache line because other computing nodes do not have the relative cache line. Therefore, in this study, we adopt the write-back invalidation protocol for our designing of the shared memory architecture in $\lambda$ computing environment. The write-back invalidation protocol is simply explained below.

In the write-back invalidation protocol, data in the local cache has three sates; Invalid (I), Clean (C) and Dirty (D). The I state means that data is invalidated and can not 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 the shared memory. We show the state transition diagram of the basic write-back invalidation protocol in Fig. 6.

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 Figure 7 (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 line does not need cache coherency 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 shared media. Since the cache controllers of other processors snoop a shared media, they receive the control message and invalidate the relevant data in their local cache (Fig. 7 (b)). Henceforth, read and write accesses to the data in a D state do not need cache coherency 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 coherency is completed. Next, the data is sent by a shared media to the processor that requested the read access and states of cache lines on both processor's cache will become C states (Fig. 7 (c1)). On the other hand, when other processor writes to the

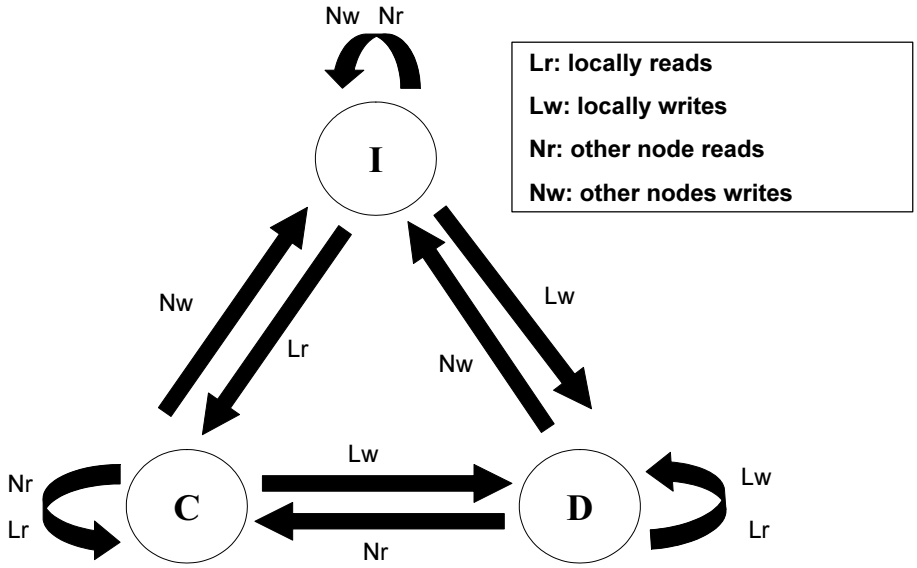



Figure 6: State transition diagram of the basic write-back invalidation protocol.

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 is invalidated (Fig. 7 (c2)).

### 2.3.4 Realization method for cache coherency protocol

In this study, we presume that each computing node has a cache, it is necessary to fully take into consideration of the coherency between the data on the cache and the data on the shared memory. Two ways, a snoop method and a directory method, are techniques for generally maintaining cache coherency.

- Snoop method

  Every cache that has a copy of the data from a line of physical memory also has a copy of the sharing status of the line, and no centralized state is kept. The caches are usually on a shared memory bus, and all cache controllers monitor or snoop on the bus to determine whether or not they have a copy of a line that is request on the bus.

- Directory method

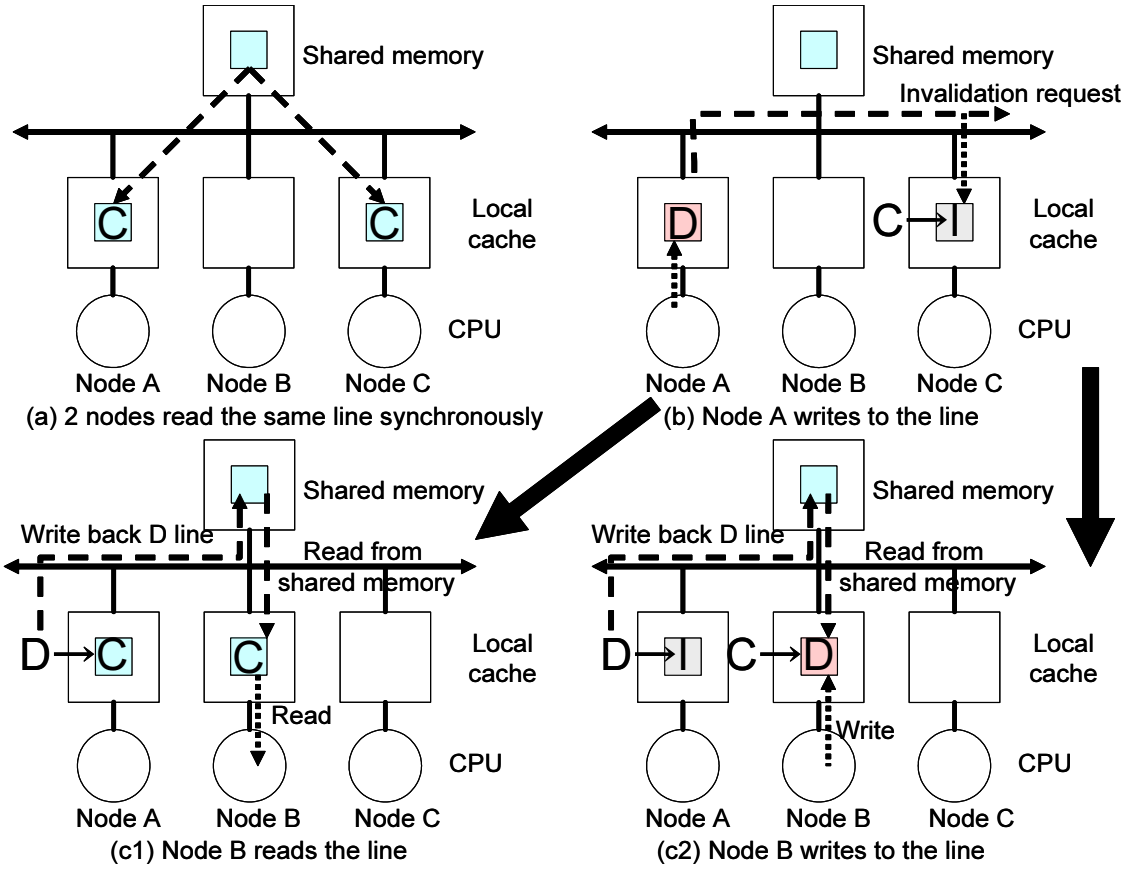



Figure 7: Behavior of the write-back invalidation protocol.

The sharing status of a line of physical memory is kept in just one location, called the directory. Information in the directory includes which caches have copies of the line, whether it is dirty, and so on. When we have to keep cache coherency, by referring this directory, invalidation signal is directly sent to target cache.

### 2.4 Proposed shared memory architecture

We propose three typical shared memory architectures listed below.

- Ring-UMA architecture (Fig. 8)

  This architecture organizes ring topology, and adopts UMA type memory access model. Moreover, we adopt the snoop method to this architecture to realize cache coherency by assuming another ring for cache control.

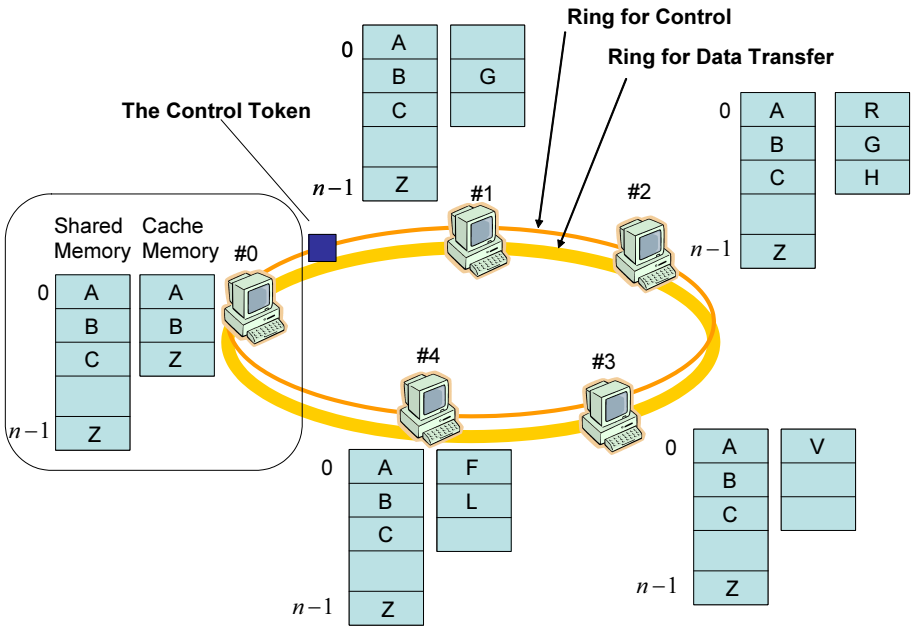



Figure 8: Ring-UMA architecture.

- Ring-NUMA architecture (Fig. 9)

  This architecture has similar characteristic to the Ring-UMA architecture. Difference between Ring-UMA architecture and Ring-NUMA architecture is memory access model. This architecture enables us to use larger address space than Ring-UMA architecture.

- Mesh-NUMA architecture (Fig. 10)

  This architecture organizes mesh topology, and adopts NUMA type memory access model. In this architecture, the directory method is used for cache coherency operation because it is difficult to realize the snoop method on mesh topology.

It is important to keep coherency between each processors and to maintain consistency between the cache memory and the shared memory for a shared memory architecture. In addition, from a parallel application level point of view, synchronization between processes is also important. Broadcasting is mainly used for these cache coherency processing and process synchronization. Then, first of all, we analyze and evaluate shared memory architecture that organizes ring topology which makes it easy to broadcast among computing nodes in $\lambda$ computing environment. In this topology, we prepare the wavelength for control on its wavelength. By snooping this token,

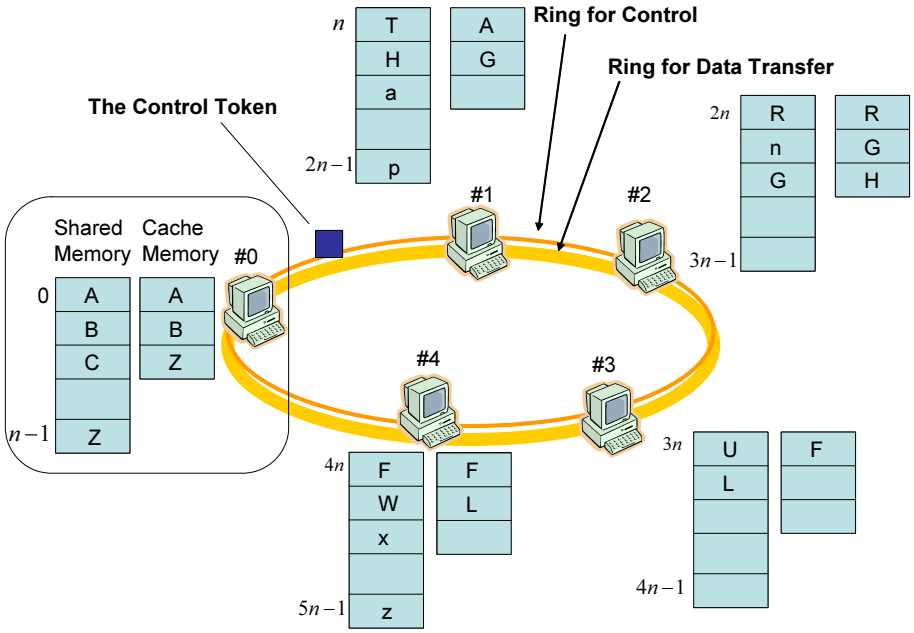



Figure 9: Ring-NUMA architecture.

we can naturally extend conventional snoop cache method. We adopt the write-back invalidation protocol to these architecture.

However, as mentioned before, communication between computing nodes must round optical ring at least one time and the rounding time becomes propagation delay. Therefore, we also study a shared memory architecture that organizes mesh topology where the average propagation delay between computing nodes become shorter than ring topology in general. In this architecture, it is difficult to prepare the wavelength for control like ring topology. Therefore, we adopt directory method for realizing cache coherency protocol. We also adopt the write-back invalidation protocol. Moreover, the processing load on each computing node is higher than that of ring topology because duplicating data frame is required on each computing node for realizing broadcasting. Therefore, NUMA type memory access model is appropriate for architectures which organize mesh topology because each computing node is responsible for a certain range of address spaces and this removes necessity of broadcasting from architectures which is adopted this topology when cache coherency processing is executed. When a computing node updates the data on its cache memory, cache coherency is done among computing nodes which has relative cache line and which is responsible for the relative physical address of the shared memory.

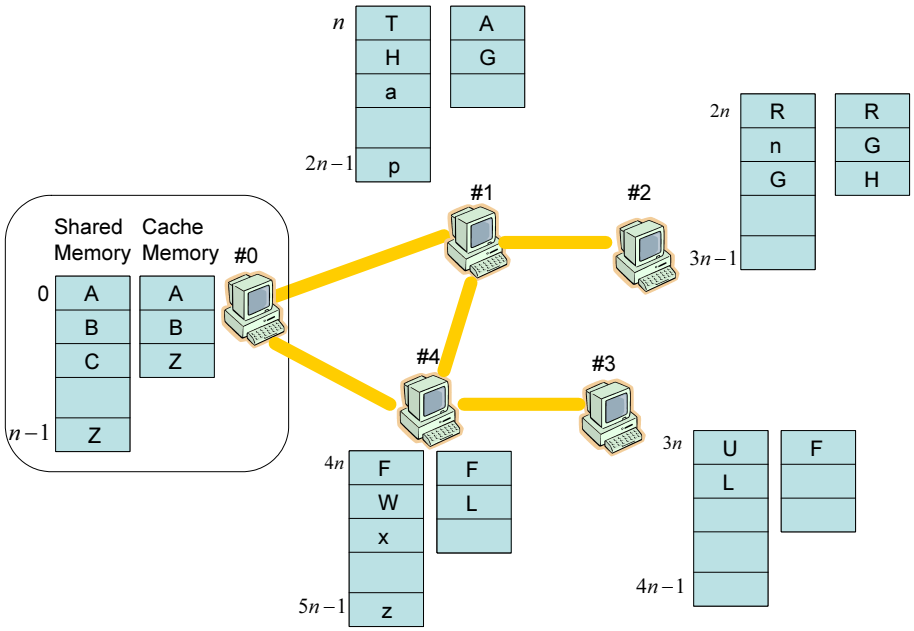



Figure 10: Mesh-NUMA architecture.

## 3 Design of shared memory architecture in $\lambda$ computing environment

In this section, we introduce three shared memory architectures which we adopt in $\lambda$ computing environment, and explain about their behavior. At first, we describe our network model. We also explain about interaction between network and computer in $\lambda$ computing environment. Next, we explain about behavior of shared memory architectures in the point of characteristic factor we described in section 2.

### 3.1 Specification of network and computing node

We show a configuration of each computing node and data flows through a network interface between a computing node and wavelengths of photonic network (Fig. 11). Computing nodes that compose $\lambda$ computing environment are connected to photonic network with optical fibers. In this study, we presuppose that each computing node has one CPU with the CPU-cache, level-2 cache memory, and a main memory. A main memory is separated into two area. One is local memory area. A local memory is used for storage of programming code and private data. Another one is shared memory area. This area is used for storage of shared data. By using this area, data sharing between computing nodes is done. It has also two types of L2 cache memory. One is for local memory which is used for caching local memory. We call this cache memory local cache. Another one is for shared memory which is used for caching shared memory. We call this cache memory shared cache. We adopt cache coherency protocol to level-2 cache not to CPU-cache. Capacity of L2 cache is $C$ KB and cache line size is $l$ KB. Therefore, the number of cache line in the shared cache is $\frac{C}{l}$.

Next, we explain about network. The bandwidth of photonic network is $B$ Gbps and the propagation delay time is 5 $\mu$s/km. For example, in the case of the distance between two computing nodes are $L$ km, it takes $5 \times L + \frac{l \times 8 \times 10^3}{B \times 10^9}$ $\mu$s to transmit one cache line. There is a cache controller at a network interface. The cache controller works for keeping the cache coherency and memory consistency by invalidating a cache line, watching the shared cache and the shared memory, managing the state of a cache line, sending/receiving a control message, updating the shared memory, responding to other computing node's message, and so on. The cache controller also works when the CPU accesses to the shared cache and shared memory. Therefore, the cache controller performs very important part for the behavior of the shared memory architecture.

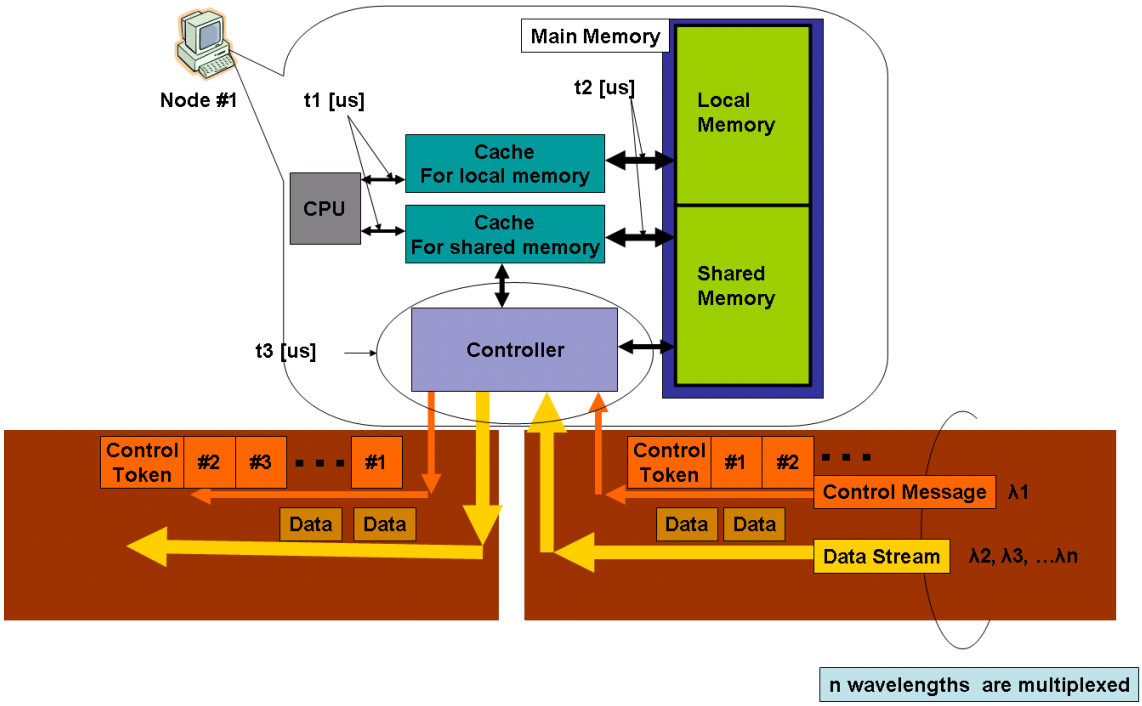



Figure 11: Network interface.

### 3.2 Design of control message

Next, we explain about control messages. We use these messages to control the behavior of a shared memory architecture. The behavior of architecture is shown in Section 3.3.

**Lock message** A computing node that receives this message prevents the locked cache line from accessing.

**Verify message** After a computing node that has the requested cache line receives this message, it attaches the state message to this message.

**State message** This message contains about the state of requested cache line.

**Copy message** A computing node that receives this message has to send the requested cache line to the computing node which sends this message.

**Write back message** A computing node that receives this message writes back the requested cache line to the shared memory.

Table 1: Parameters of shared memory architecture.

| | | |
|---|---|---|
| Transmission speed in a optical fiber | 5 | [$\mu$s/km] |
| Access time between CPU–L2 cache | $t_1$ | [$\mu$s] |
| Access time between L2 cache–Main memory | $t_2$ | [$\mu$s] |
| Frame processing time at network interface | $t_3$ | [$\mu$s] |
| Capacity of shared memory of each computing nodes | $M$ | [MB] |
| Capacity of L2 cache memory | $C$ | [KB] |
| Cache line size | $l$ | [KB] |
| Bandwidth of network | $B$ | [Gbps] |
| Number of node | $N$ | |

**Invalidation message** Computing nodes that receive this message invalidate the relevant cache line which is on own shared cache.

### 3.3 Design of function and behavior

Now, we describe the design of shared memory architectures by mainly focusing on its function and behavior. We adopt the write-back invalidation protocol for cache coherency to all architecture. Because, this protocol is often adopted to parallel computing system. In a system which adopts this protocol, there is only one CPU which has the latest cache line by invalidating other CPU's relevant cache line. This protocol realize cache coherency by this simple mechanism. This protocol has also the least shared bus or network usage. So, this protocol is suitable for $\lambda$ computing environment.

Before we explain about behavior of these architectures, we define some terminology. These terminologies are for Ring-NUMA architecture and Mesh-NUMA architecture.

- **Home memory**

  Home memory is a part of the shared memory that each computing node is responsible for.

- **Home node**

  Home node is a computing node that is responsible for the own shared memory and cache coherency between home memory and other shared cache.

- **Ownership**

  Ownership is the right to update the cache line.

### 3.3.1 Ring-UMA architecture

This architecture organizes ring topology, and makes it easier to broadcast data and to realize synchronization among computing nodes which compose $\lambda$ computing environment. However, when a computing node sends data, it requires at least the time to go round once on an optical ring as propagation delay. This factor may influence the performance of this architecture. This architecture has UMA type memory access model. So, the shared memory of each computing node has same address space and then computing nodes share all of data which is on the shared memory (See Fig. 8). Therefore, total capacity of the shared memory is $M$ GB. In this architecture, each computing node does not require to use network when they access to shared data. They only have to access to own shared memory because all of shared data exist on own shared memory. We provide a wavelength to realize the snooping method for the cache coherency. We explain about behavior of this architecture below.

**Behavior of read access** In the case of read access and the cache line is

$C$ **or** $D$ **state:** Each computing node has only to access to the shared cache and reads data. The processing of cache coherency protocol is not needed.

$I$ **state:** Required data does not exist on the shared cache. So, computing nodes have to access to the shared memory to get the data. However, other computing nodes may have the latest data. Therefore, computing nodes have to process the cache coherency protocol as follows.

(1) A computing node (node A) waits for the control token.

(2) After receiving the control token, node A attaches the verify message to the token and sends it to other computing nodes.

(3) A computing node (node B) which has the relevant cache line has to respond to the verify message and attaches the state message.

(4) After going round on the optical ring, node A can receive the state message. If the state message is attached to the control token, there is $C$ or $D$ state cache line in other

shared cache. If the state message is not attached, the relevant cache line does not exist in other shared cache and this is same as in $I$ state. If the state of other cache is

$I$ **or** $C$**state:** Node A only has to copy the cache line from the shared memory and read data. Then the state of the cache line is changed to $C$ state. The cache coherency protocol is not needed.

$D$ **state:** Node A attaches the write back message to the control token. Node B, which has the $D$ state cache line, must respond this message, send back the cache line to the shared memory, and change the state of this cache line to $C$ state. Then, node A copies the received data to the shared cache, sets the state of cache line to $C$ state, and reads data from cache.

**Behavior of write access** In the case of write access and the cache line is

$C$ **state:** Each computing node can access to the shared cache and update data. Invalidating other shared cache is needed.

$D$ **state:** Each computing node also has only to access to the shared cache and updates data. Invalidating other shared cache is not needed.

$I$ **state:** Required data does not exist on the shared cache. So, computing nodes have to access to the shared memory to get the data. However, other computing nodes may have the latest data. Therefore, computing nodes have to process the cache coherency protocol as follows.

(1) A computing node (node A) waits for the control token.

(2) After receiving the control token, node A attaches the verify message to the token and sends it to other computing nodes.

(3) A computing node (node B) which has the relevant cache line has to respond to the message and attaches the state message.

(4) After going round on the optical ring, node A can receive the state message. If the state message is attached to the control token, there is $C$ or $D$ state cache line in other shared cache. If the state message is not attached, the relevant cache line does not

exist in other shared cache and this is same as in $I$ state. If the cache line of other shared cache is

$I$ **state** Node A copies the received cache line to the shared cache and updates data. Then node A changes the state of the cache line to $D$ state.

$C$ **state:** Node A updates the cache line. Then, invalidating other cache is performed.

$D$ **state:** Node A attaches the write back message to the control token. Node B, which has the $D$ state cache line, must respond this message, write back the cache line to the shared memory and change the state of this cache line to $C$ state. Then, node A copies the received cache line to the shared cache and updates the data. Then invalidating other cache is performed.

**Behavior of invalidating other cache** Invalidating other shared cache and keeping cache coherency is realized as follows.

(1) Node A waits for the control token.

(2) Node A attaches the lock message to the token. Other computing nodes are prevented from accessing to the relevant cache line.

(3) After going round on the optical ring, node A updates the cache line, changes the state to $D$ state and sends invalidation signal.

(4) After finishing invalidation, node A attaches the unlock message to the control token.

**Write back the cache line to the shared memory**

(1) Requested computing node attaches the lock message to the control token when it receives the write back message.

(2) After going round on the optical ring, it writes back the cache line to the shared memory. Then the state of the cache is changed to $C$ state.

(3) Then the unlock message is attached to the control token.

### 3.3.2 Ring-NUMA architecture

This architecture organizes ring topology, so Ring-NUMA architecture has similar characteristics to Ring-UMA architecture. However, Ring-NUMA architecture has NUMA type memory access model. Therefore, each computing node has a part of address space. Then, by merging each shared memory, we can built one shared memory, that has $M \times N$ GB capacity, and share whole address space (See Fig. 9). A computing node may have to use network when they access to shared data. It can access to the shared memory without using network when the requested data exists on own shared memory. However, if the requested data exists on other shared memory, computing nodes access to other shared memory through network. We also adopt the snooping method for cache coherency. We explain about behavior of this architecture below. The behavior of this architecture is similar to Ring-UMA architecture.

**Behavior of read access** In the case of read access and the cache line is

$C$ **or** $D$ **state:** Each computing node has only to access to the shared cache and reads data. The processing of cache coherency protocol is not needed.

$I$ **state:** Required data does not exist on the shared cache. So, computing nodes have to access to the shared memory to get the data. However, other computing nodes may have the latest data. Therefore, computing nodes have to process the cache coherency protocol as follows.

(1) A computing node (node A) waits for the control token.

(2) After receiving the control token, node A attaches the verify message to the token and sends it to other computing nodes.

(3) A computing node (node B) which has the relevant cache line has to respond to the message and attaches the state message.

(4) After going round on the optical ring, node A can receive the state message. If the state message is attached to the control token, there is $C$ or $D$ state cache line in other shared cache. If the state message is not attached, the relevant cache line does not exist in other shared cache and this is same as in $I$ state. If the state of other cache is

$I$ **or** $C$ **state:** Node A sends the copy message to home node. Home node sends the cache line to node A. Then node A copies the received cache line to the shared cache and reads data. Then the state of the cache line is changed to $C$ state.

$D$ **state:** Node A attaches the write back message to the control token. Node B, which has the $D$ state cache line, must respond this message and sends back the cache line to the home node and changes the state of this cache line to $C$ state. Home node writes back the received cache line to the home memory. Then, home node sends the cache line to node A. Node A copies the cache line to the shared cache, sets the state of cache line to $C$ state, and reads data from the shared cache.

**Behavior of write access** In the case of write access and the cache line is

$C$ **state:** Each computing node can access to the shared cache and update the cache line. Invalidating other cache is needed.

$D$ **state:** Each computing node also has only to access to the shared cache and update the cache line. Invalidating other shared cache is not needed.

$I$ **state:** Required data does not exist on the shared cache. So, computing nodes have to access to the shared memory to get the data. However, other computing nodes may have the latest data. Therefore, computing nodes have to process the cache coherency protocol as follows.

(1) A computing node (node A) waits for the control token.

(2) After receiving the control token, node A attaches the verify message to the control token and sends it to other computing nodes.

(3) A computing node (node B) which has the relevant cache line has to respond to the message and attaches the state message.

(4) After going round on the optical ring, node A can receive the state message. If the state message is attached to the control token, there is $C$ or $D$ state cache line in other shared cache. If the state message is not attached, the relevant cache line does not exist in other shared cache and this is same as in $I$ state. If the state of other cache is

$I$ **state** Node A sends the copy message to home node. Home node sends the cache line to node A. Then node A copies the received cache line to the shared cache and writes data. Then the state of the cache line is changed to $D$ state.

$C$ **state:** Node A sends the copy message to home node. Home node sends the cache line to node A. Then node A copies the received cache line to the shared cache and updates data. Invalidating other shared cache is performed.

$D$ **state:** Node A attaches the write back message to the control token. Node B, which has the $D$ state cache line, must respond this message and then sends back the cache line to the home node and changes the state of this cache line to $C$ state. Then, home node receives the cache line, updates home memory, and sends the cache line to node A. Then, invalidating other cache is performed.

**Invalidating other cache** The behavior of invalidating other cache is same as Ring-UMA architecture and we already described about this. So, we omit to explain about this process.

### 3.3.3 Mesh-NUMA architecture

This architecture organizes mesh topology and makes average propagation delay between computing nodes shorter than that of ring topology. In this architecture, we adopt the directory method for realizing the cache coherency protocol.

We introduce three states of the shared memory.

- $U$ state

  This state means that a block which is relevant to each cache line on home memory is not cached.

- $S$ state

  This state means that a copy that is consistent with home memory exists on other shared cache.

- $D$ state

  This state means that a copy that is not consistent with home memory exists on other shared cache.

By using these states, we describe about behavior of this architecture below.

**Behavior of read access** In the case of read access and the cache line is

$C$ **or** $D$ **state:** Each computing node has only to access to the shared cache and read data. In this case, the processing of cache coherency protocol is not needed.

$I$ **state:** Required data does not exist on the shared cache. So, computing nodes have to access to the shared memory to get the data. However, in this case, other computing nodes may have the latest data. Therefore, computing nodes have to process the cache coherency protocol as follows.

(1) A computing node (node A) requests to home node to copy the block of home memory which contains required data.

(2) The home node verifies the state of requested block. If the state of block on home memory is

$U$ **or** $S$ **state:** Home node sends back the requested block and sets the state of block on the shared memory to $S$ state.

$D$ **state:** Home node looks up directory and searches the computing node (node B) which has the latest cache line. Then, the write back message is sent by home node to node B. After node B writes back the cache line, home node sends the requested block to node A and sets the state of block on the shared memory to $S$ state. Node A receives the latest data block, copies to the shared cache, sets the cache line state to $C$ state and reads data.

**Behavior of write access** In the case of write access and the cache line is

$D$ **state:** Each computing node has only to access to the shared cache and read data. The processing of cache coherency protocol is not needed.

$C$ **state:** Other computing node may have the relevant cache line. So, invalidating other shared cache is needed. After invalidating other shared cache, the state of cache line is set to $D$ state and updated.

$I$ **state:** Required data does not exist on the shared cache. So, computing nodes have to access to the shared memory to get the data. However, in this case, other computing nodes may have

the latest data. Therefore, computing nodes have to process the cache coherency protocol as follows.

(1) A computing node (node A) requests to home node to copy the block of home memory which contains required data.

(2) A home node verifies the state of requested block. If the state of block is

$U$ **state:** Home node sends back the requested block and sets the state of block on the shared memory to $S$ state.

$S$ **state:** Other computing node may have the relevant cache line. So, invalidating other shared cache is needed. After invalidating other shared cache, node A receives the cache line from home node. Then node A sets the state of cache line to $D$ state and updates the data.

$D$ **state:** Home node looks up directory and searches the computing node (node B) which has the latest cache line. Then, write back request is sent by home node to node B. After node B writes back the cache line, home node sends the requested block and sets the state of block on the shared memory to $D$ state. Node A receives the latest data block, sets the cache line state to $D$ state and writes data.

**Invalidating other cache** Invalidation of other cache is processed as follows.

(1) A computing node (node A) requests ownership to home node.

(2) Home node looks up directory and searches computing nodes that have the relevant cache line.

(3) Home node sends invalidation signal to other computing nodes that have the relevant cache line. After that, home node waits for receiving Ack.

(4) Computing nodes that receive invalidation signal invalidate the relevant cache line and set the state to $I$ state. Then, they send Ack to home node.

(5) After receiving Ack, home node transfers the ownership and the cache line to node A. Then, the state of relevant home memory block is set to $D$ state.

## 4 Modeling and analysis with semi-Markov process

In this section, we model and analyze the shared memory architecture we designed in section 3. For modeling and analyzing, we utilize semi-Markov process.

### 4.1 Semi-Markov process

In this study, we use semi-Markov process [7] for modeling the shared memory architecture. In semi-Markov process, we can voluntarily set the residence time of each state. Therefore, we consider that semi-Markov process is suitable for modeling the shared memory architecture where a complicated request such as cache coherency control arises. We describe the definition of semi-Markov process [8] below.

**Definition:** Let $\{X(t), t \geq 0\}$ be the stochastic process that has state space which is composed by countable set. In $\{X(t)\}$, let the moments of the state changes be $t_0 < t_1 < t_2 \cdots$ and put $X_n = X(t_n)$. In this case, we call the stochastic process $\{X(t), t \geq 0\}$, that $\{X_n | n = 0, 1, 2, \cdots\}$ forms Markov process, as semi-Markov process.

To obtain the steady state probability for semi-Markov process, we can follow the same method to solve the steady state probabilities for discrete time Markov chains. Indeed, at the moments of state changes, the semi-Markov process behaves exactly as a discrete time Markov chains. Concrete algorithm to obtain the steady state probabilities for semi-Markov process is as follows [7, 9].

(1) Compute the steady state probability for the discrete time Markov chain with state transition matrix $p = (p_{i,j})$, denoted here as $\pi$.

(2) Compute the average state residence times $\eta_i$ for all state $i$ in the semi-Markov process.

(3) Compute the steady state probability in the semi-Markov process by taking these residence time into account, as follows.

$$P_i = \frac{\pi_i \eta_i}{\sum_j \pi_j \eta_j} \tag{1}$$

We can also obtain the escape probabilities $\lambda_i$ as follows.

$$\lambda_i = \frac{P_i}{\eta_i} \tag{2}$$

Table 2: Given parameters for models.

| | |
|---|---|
| Cache hit ratio | $h$ |
| Ratio of read access per main memory access | $r$ |
| Ratio of write access per main memory access | $w$ |
| Ratio of shared memory access per main memory access | $s$ |

### 4.2 Variable definition in model

We describe variables with given parameters in Table 2.

The probability $P_D$ that a certain computing node has a certain cache line in $D$ state is expressed as follows.

$$P_D = hws \tag{3}$$

Then the probability $P_d$ that only one of other computing nodes has a cache line in $D$ state and other computing nodes do not have the cache line in $D$ state is expressed as follows.

$$P_d = (N-1)hws(1-hws)^{N-1} \tag{4}$$

The probability $P_C$ that a certain computing node has a certain cache line in $C$ state is expressed as follows.

$$P_C = hrs \tag{5}$$

Then the probability $P_c$ that at least one computing node except for itself has a cache line in $C$ state is expressed as follows.

$$P_c = 1 - (1-hrs)^{N-1} \tag{6}$$

The probability $P_x$ that there is no space for a new cache line is considered as follows. When the cache memory has no space for the new cache line, an open space for new line is made by invalidation signal from other computing nodes. So, the probability $P_x$ is expressed as follows.

$$P_x = (1-P_{inv})^{N-1} \tag{7}$$

where $P_{inv}$ is the probability that a certain cache line is invalidated by other computing nodes. The invalidation signal is sent when a certain computing node does write access to the relevant

cache line or does write access to the cache line which is not in its cache memory and also not in $D$ state. Therefore, $P_{inv}$ is expressed as follows.

$$P_{inv} = hP_c w + (1-h)(1-P_d)w \tag{8}$$

### 4.3 Modeling of shared memory architecture

We model shared memory architecture in $\lambda$ computing environment by using semi-Markov process. We make the state transition diagram in the point of view of CPU on each computing node. In order to make the state transition diagram, we follow the behavior of each architecture we designed in section 3.

#### 4.3.1 Ring-UMA architecture

Fig. 12 shows the state transition diagram of the Ring-UMA architecture. The state 1 is computation state. That is, in this state, a CPU can execute instructions without any memory accesses. Then, if a LOAD or STORE instruction is executed, the state is change to the state 2 or state 32. In the state 32, a CPU accesses to the local memory, then changes the state to 1. When the state changes to the state 2, it immediately changes to one of the state $\{3, 4, 8, 22, 23, 31\}$ because this state is transient state. In the case of read access, the state changes to $\{3, 4, 8\}$, and in the case of write access, the state changes to $\{22, 23, 31\}$. That is, the state changes according to the behavior we designed in section 3.3.1.

An access to the shared memory can be blocked when other computing nodes are accessing to the same cache line. We can obtain this probability $P_B$ as follows. We presume that each CPU access the cache line in random, so each cache line is accessed by $\frac{l}{C}$. Moreover, the set of the state that a CPU is accessing to the cache line are $\{10, 19, 20, 21, 25, 34\}$, let this set be $S_a$. So, an access is blocked when at least one computing node tries to access the same line and the state is at $s \in S_a$. Therefore, the probability $\alpha_1$ that each cache line is accessed is expressed as follows.

$$\alpha_1 = \frac{l}{C} \sum_{s \in S_a} P_s \tag{9}$$

where $S_a = \{10, 19, 20, 21, 25, 34\}$. Therefore, $P_B$ is expressed as follows.

$$P_B = 1 - (1-\alpha_1)^{N-1} \tag{10}$$

A CPU must write back the cache line to the shared memory when other computing nodes send the write back message. We can obtain this probability $P_q$ as follows. The write back message is sent when at least one computing node leave the state 10 or 25 by probability $\lambda_{10}$ or $\lambda_{25}$ and change the state to 14 or 26 by $P_d$. So, the probability $P_q$ is expressed as follows.

$$P_q = P_d(\lambda_{10} + \lambda_{25}) \tag{11}$$

### 4.3.2 Ring-NUMA architecture

Fig. 13 shows the state transition diagram of the Ring-NUMA architecture. Most of the state transition and transition probabilities is same as the Ring-UMA architecture. So, we do not explain about this architecture in detail. The important thing in the state transition diagram of Ring-NUMA architecture is the residence time of state $\{15, 17, 32\}$. When a cache miss occurs, these states are sure to transit, and at this state, the copy message is sent to home node. If the node that sends the copy message is the home node, it only has to access to own shared memory. This probability is $\frac{M}{M \times N} = \frac{1}{N}$. If the node that sends the copy message is not the home node, it has to wait for receiving the cache line from the home node by probability $1 - \frac{1}{N}$. Therefore, the residence time of the state $\{15, 17, 32\}$ is obtained as follows.

$$\eta_{15} = \eta_{17} = \eta_{32} = t_2 \frac{1}{N} + \tau(1 - \frac{1}{N}) \tag{12}$$

where $\tau$ is the time to round the ring once.

Moreover, we can also obtain the probability $P_B$ and $P_q$ in the same way described in section 4.3.1.

$$\alpha_2 = \frac{l}{C} \sum_{s \in S_b} P_s \tag{13}$$

where $S_b = \{10, 15, 17, 21, 22, 23, 27, 32, 37\}$ in Fig. 13. Therefore, $P_B$ is expressed as follows.

$$P_B = 1 - (1 - \alpha_2)^{N-1} \tag{14}$$

Then $P_q$ is expressed as follows.

$$P_q = P_d(\lambda_{10} + \lambda_{27}) \tag{15}$$

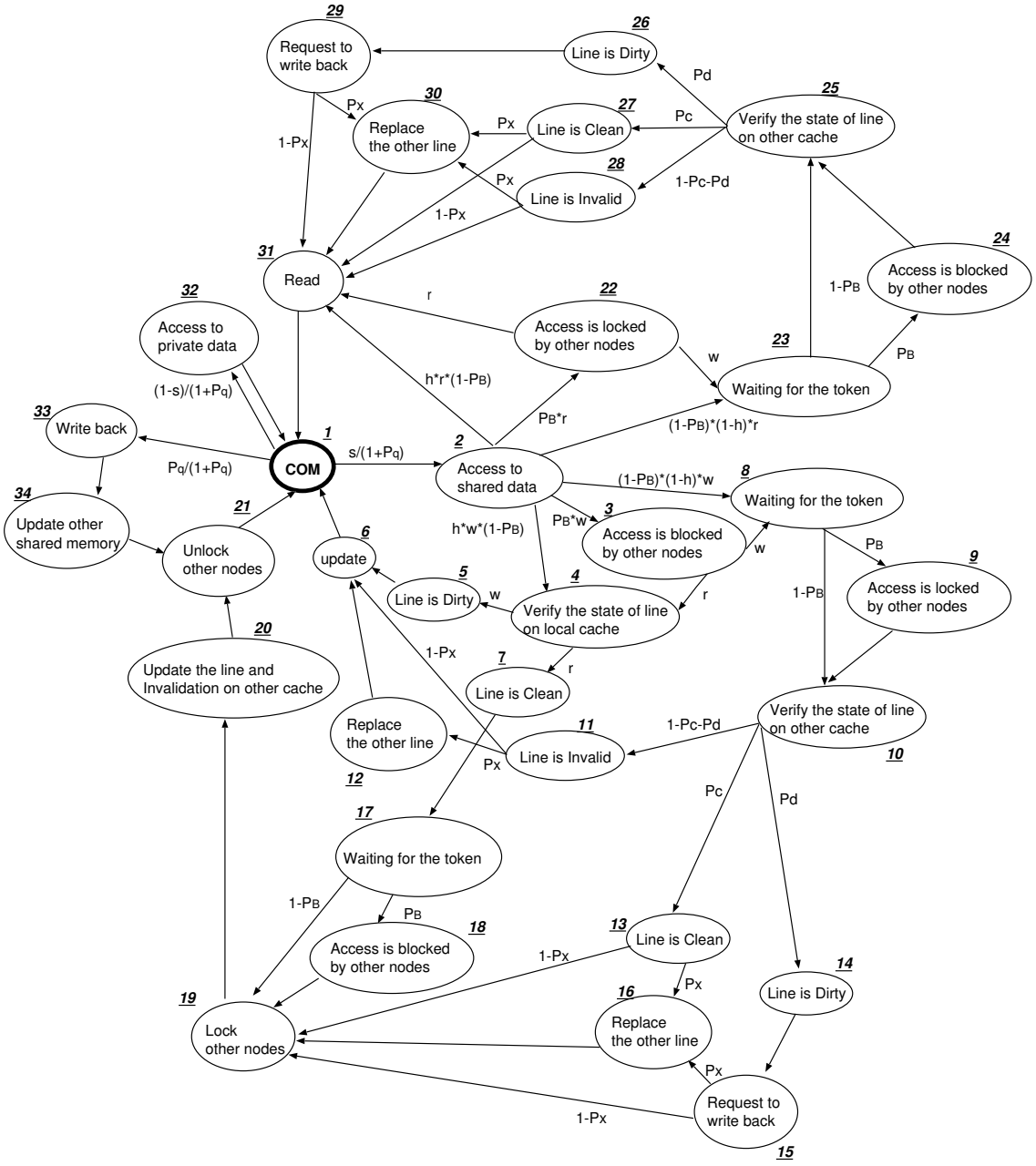



Figure 12: State transition diagram of Ring-UMA architecture.

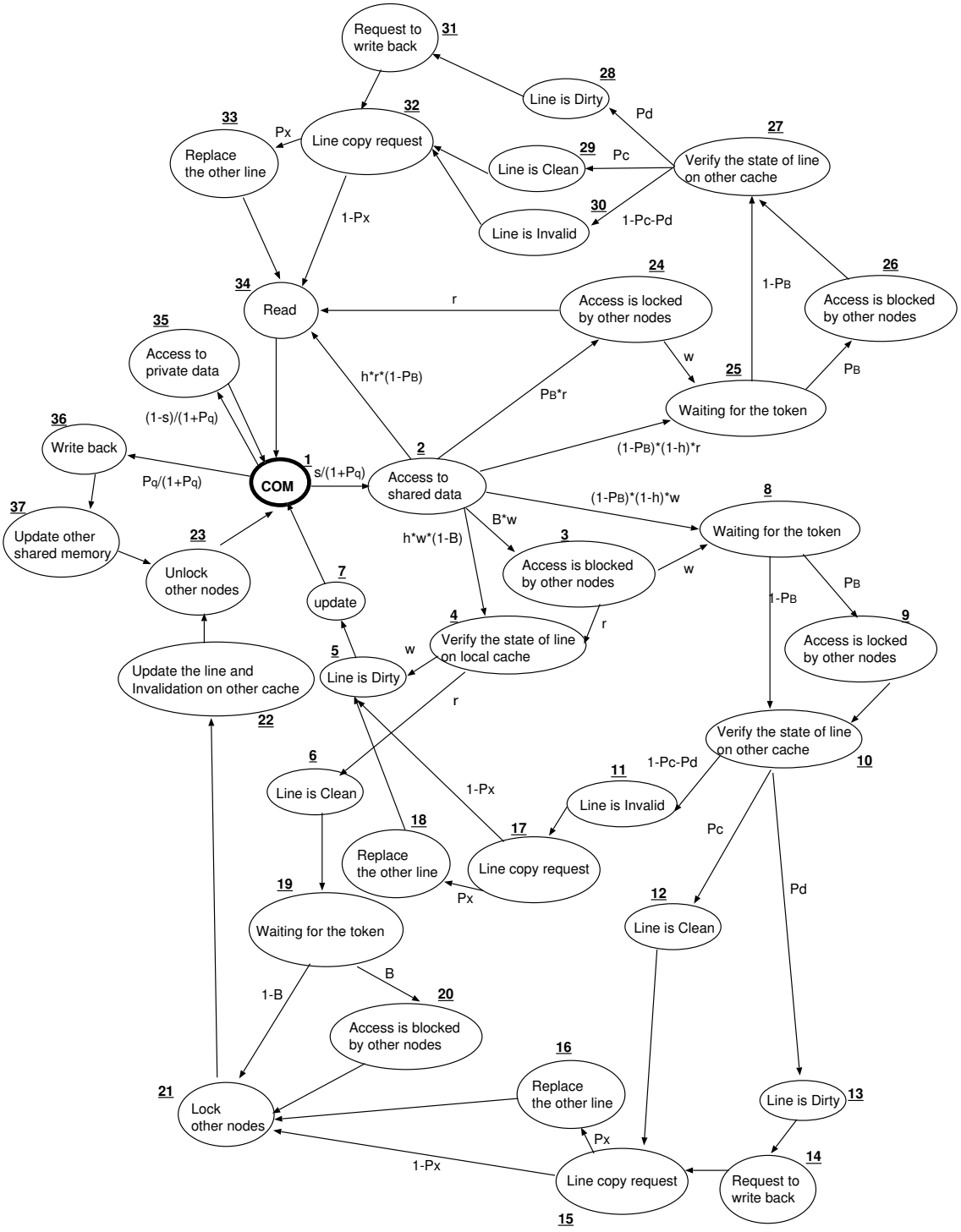



Figure 13: State transition diagram of Ring-NUMA architecture.

### 4.3.3 Mesh-NUMA architecture

Fig. 13 shows the state transition diagram of the Mesh-NUMA architecture. In this diagram, the state 1 is also computation state and if a LOAD or a STORE instruction is executed, then state is change to the state 2. Then the state changes according to the behavior we designed in section 3.3.3. When the state reaches at the state $\{4, 14\}$, the copy message or the invalidation message is sent to a home node and enqueued by the probability $\alpha_3$,

$$\alpha_3 = (P_c + P_d)(\lambda_4 + \lambda_{14}). \tag{16}$$

Therefore, when a message is sent to the home node, the average number of messages $\overline{K}$ in the queue is derived as

$$\overline{K} = \sum_{i=1}^{N-1} i \times ({}_{N-1}C_i {\alpha_3}^i (1-\alpha_3)^{N-1-i}), \tag{17}$$

and the average waiting time in the queue $\bar{t}_q$ is expressed as follows.

$$\begin{aligned} \bar{t}_q &= \overline{K} \times (\eta_{26} + \eta_{27}) \\ &= \overline{K} \times (\eta_{29} + \eta_{30}) \\ &= \overline{K} \times 2\tau \end{aligned} \tag{18}$$

Therefore, the residence time at the state $\{4, 14\}$ is derived as

$$\eta_4 = \eta_{14} = \tau + \bar{t}_q \tag{19}$$

where $\tau$ is propagation delay between the home node and the node which sends the message.

When a node has the cache line in $C$ state and the home node sends the invalidation message to this node by the probability $\lambda_{29}$. Therefore, the probability $P_A$ that this node receives the invalidation message is expressed as follows,

$$P_A = P_C \lambda_{29}. \tag{20}$$

When a node has the cache line in $D$ state and the other node accesses to the relative cache line, the home node sends the invalidation message to this node by the probability $\lambda_{26}$. Therefore, the probability $P_E$ that this node receives the write back message is expressed as follows,

$$P_E = P_D \lambda_{26}. \tag{21}$$

When at least one node leaves the state $\{4, 14\}$ and a cache line is $C$ or $D$ state in other shared cache, the home node receives the copy message. Therefore, the probability $P_G$ that the home node looks up directory for the cache coherency protocol processing is derived as,

$$P_G = 1 - (1 - (P_C + P_D)(\lambda_4 + \lambda_{14}))^{N-1}. \tag{22}$$

Lastly, we show the residence time table at Table. 3

Table 3: The residence time table [$\mu$s].

| State | Ring-UMA | Ring-NUMA | Mesh-NUMA |
|---|---|---|---|
| 1 | 0.002 | 0.002 | 0.002 |
| 2 | 0 | 0 | $t_1$ |
| 3 | $2\tau$ | $2\tau$ | 0 |
| 4 | $t_1$ | $t_1$ | $(1 - \frac{1}{N})\tau$ |
| 5 | 0 | 0 | $(1 - \frac{1}{N})\tau$ |
| 6 | $t_1$ | 0 | $\tau$ |
| 7 | 0 | $t_1$ | $\tau$ |
| 8 | $0.5\tau$ | $0.5\tau$ | $\tau$ |
| 9 | $2\tau$ | $2\tau$ | $\tau$ |
| 10 | $\tau$ | $\tau$ | $t_1 + t_2$ |
| 11 | 0 | 0 | $\tau$ |
| 12 | $t_2$ | 0 | $t_2$ |
| 13 | 0 | 0 | $t_1$ |
| 14 | 0 | $2\tau$ | $(1 - \frac{1}{N})\tau$ |
| 15 | $2\tau$ | $\frac{1}{N}t_2 + (1 - \frac{1}{N})\tau$ | $\tau$ |
| 16 | $t_2$ | $t_2$ | $\tau$ |
| 17 | $0.5\tau$ | $\frac{1}{N}t_2 + (1 - \frac{1}{N})\tau$ | $t_1 + t_2$ |
| 18 | $2\tau$ | $t_2$ | $\tau$ |
| 19 | $\tau$ | $0.5\tau$ | $t_2$ |
| 20 | $\tau$ | $2\tau$ | $t_1$ |
| 21 | $\tau$ | $\tau$ | $ht_1 + (1 - h)(t_1 + t_2)$ |

Table 3: The residence time table [$\mu$s].

| State | Ring-UMA | Ring-NUMA | Mesh-NUMA |
|---|---|---|---|
| 22 | $2\tau$ | $\tau$ | $t_1$ |
| 23 | $0.5\tau$ | $\tau$ | $t_3$ |
| 24 | $2\tau$ | $2\tau$ | $t_1 + t_2$ |
| 25 | $\tau$ | $0.5\tau$ | $t_1 + t_2$ |
| 26 | 0 | $2\tau$ | $\tau$ |
| 27 | 0 | $\tau$ | $\tau$ |
| 28 | 0 | 0 | $t_1 + t_2$ |
| 29 | $2\tau$ | 0 | $\tau$ |
| 30 | $t_2$ | 0 | $\tau$ |
| 31 | $t_1$ | $2\tau$ | $t_1 + t_2$ |
| 32 | $ht_1 + (1-h)(t_1 + t_2)$ | $\frac{1}{N}t_2 + (1 - \frac{1}{N})\tau$ | |
| 33 | $t_1 + t_2$ | $t_2$ | |
| 34 | $2\tau$ | $t_1$ | |
| 35 | | $ht_1 + (1-h)(t_1 + t_2)$ | |
| 36 | | $t_1 + t_2$ | |
| 37 | | $2\tau$ | |

### 4.4 Analysis by using semi-Markov process

In this section we explain about the procedure to obtain steady state probabilities for the analysis and we show some numerical results.

#### 4.4.1 Analytic approach

We cannot obtain steady state probabilities directly by solving semi-Markov models described in section 4.1. Because some of transition probabilities depend on steady state probabilities in a semi-Markov model. Therefore, we achieve these probabilities by giving an appropriate initiate value and iterating calculation until convergence. We show the procedure below.

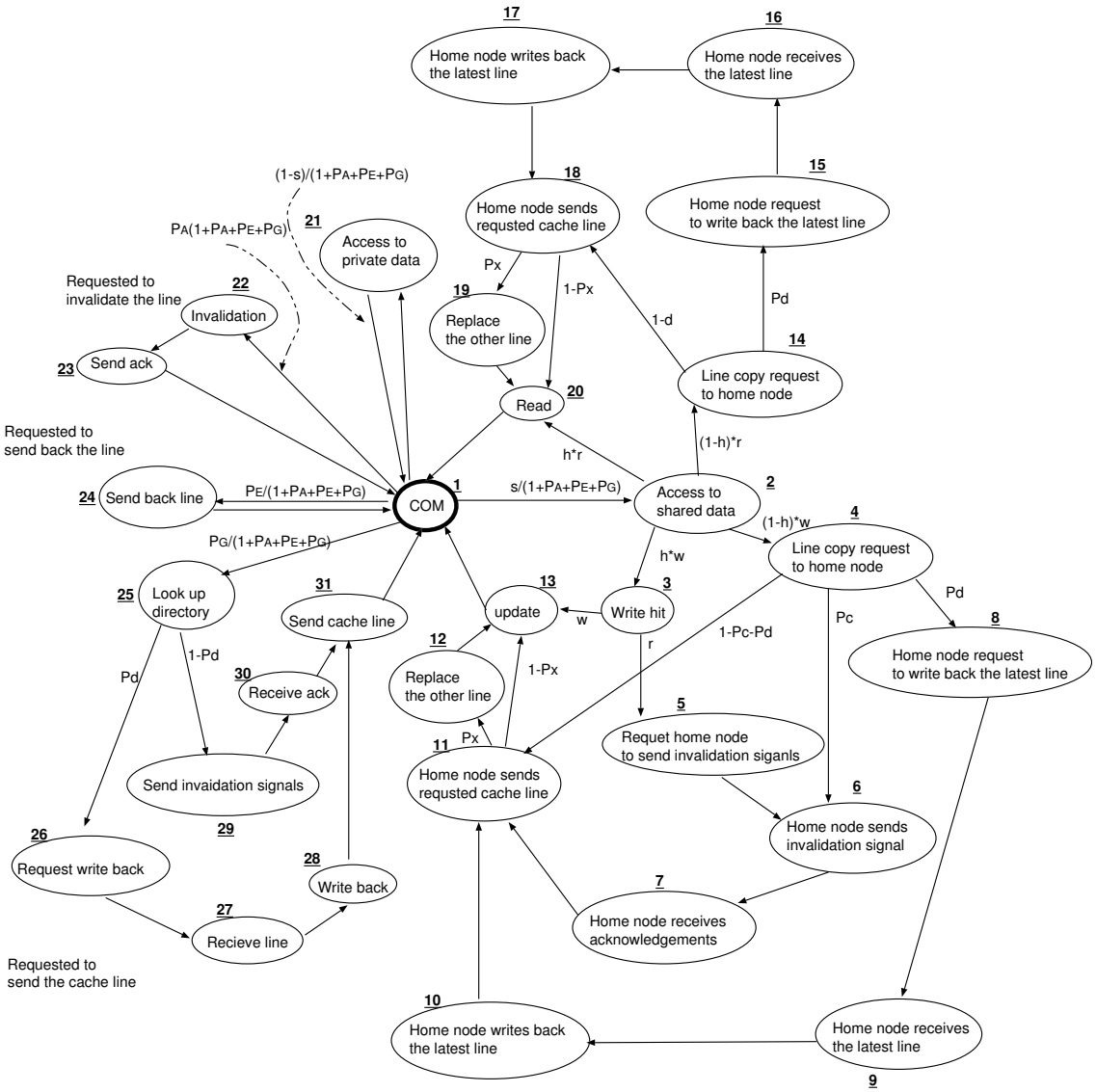
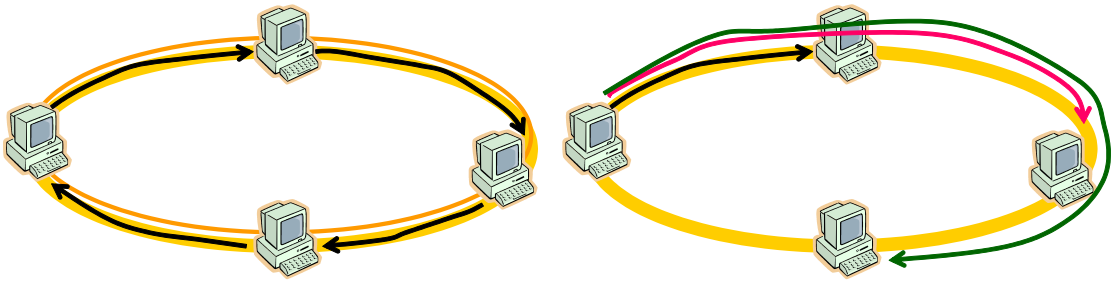



Figure 14: State transition diagram of Mesh-NUMA architecture.



(a) Ring topology (b) Mesh topology

Figure 15: Ring topology in physical

(1) Initialize transition matrix $p = (p_{i,j})$

(2) Obtain the stationary distribution vector $\pi = \{\pi_i\}$ in discrete time Markov chain by solving the equation $\pi = \pi p$

(3) Obtain steady state probabilities by expression (1)

(4) Update transition probabilities by using $\{P_i\}$

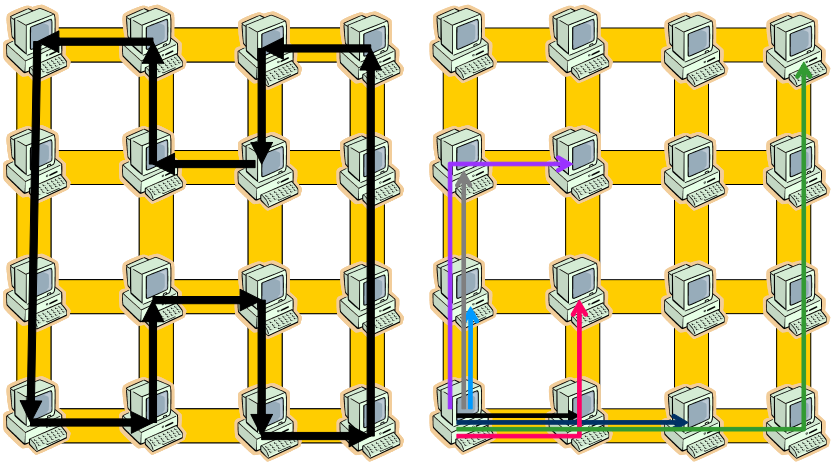
(5) Return to 2 when the difference between the latest $\{P_i\}$ and the previous $\{P_i\}$ is larger than given enough small value

### 4.4.2 Numerical analysis

We analyze our models by giving parameters appropriate values and show numerical results of each architecture. At first, we explain about two scenarios we suppose.

**Scenario 1** We built ring topology in physical on $\lambda$ computing environment, then establish ring topology (Ring-UMA architecture and Ring-NUMA architecture) or mesh topology (Mesh-NUMA architecture) in logical. In this scenario, we give the length of optical ring as parameter $L$ in advance, so the length of optical ring network is independent to the number of computing nodes.

**Scenario 2** We built grid topology in physical on $\lambda$ computing then establish ring topology (Ring-UMA architecture and Ring-NUMA architecture) or mesh topology (Mesh-NUMA archi-



(a) Ring topology (b) Mesh topology

Figure 16: Mesh topology in physical

tecture) in logical. In this scenario, we give the distance between computing nodes as parameter $L$, so the length of ring in Ring-UMA and Ring-NUMA architecture is decided by the number of computing nodes.

We also set various values in shared memory architecture for analysis. We show the values at Table. 4. Moreover, a CPU uses LOAD instruction by 15 %, STORE instruction by 5 % and other instructions by 80 %. Therefore, we set the parameter $r$ to 0.75 and $w$ to 0.25. Then, we analyze by changing the parameter $N$; the number of node, $s$; ratio of shared memory access, and $L$; length of ring (in scenario 1) or distance between computing nodes (in scenario 2).

Fig. 17 through Fig. 19 show the distributions of the steady state probabilities of Ring-UMA architecture. In Ring-UMA architecture, $P_{19}, P_{20}, P_{21}$, which are the states of processing cache coherency protocol, are the largest. As the length of ring become longer (See Fig. 17), these probabilities become larger. On the contrary, especially, $P_{32}$, which is the state of accessing to the local memory, becomes smaller and smaller. This is because the $\eta_{19}, \eta_{20}, \eta_{21}$, which are the residence time, are decided by the length of ring, and $P_{20}, P_{21}, P_{22}$ are derived by weighting each residence time. Therefore, as the length of ring becomes longer, these steady state probabilities become larger, and occupy the majority of the distribution of steady state probabilities. However, the increase of these probabilities is relatively small, at most about 10 %, on the contrary, the

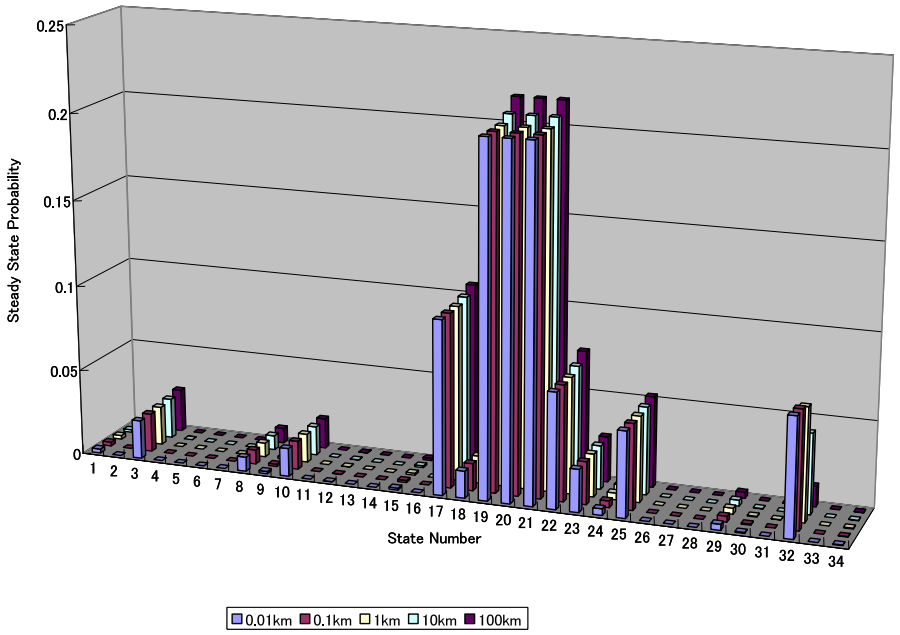
Table 4: Numerical values of parameters in models.

| | | |
|---|---|---|
| Clock frequency of a CPU | 2 | [GHz] |
| Access time between CPU–L2 cache | $0.01$ | [$\mu$s] |
| Access time between L2 cache–Main memory | $1$ | [$\mu$s] |
| Frame processing time at network interface | $3$ | [$\mu$s] |
| Capacity of shared memory of each computing nodes | $1024$ | [MB] |
| Capacity of L2 cache memory | 1024 | [KB] |
| Cache line size | 4 | [KB] |
| Bandwidth of network | 10 | [Gbps] |
| Cache hit ratio | 0.95 | |

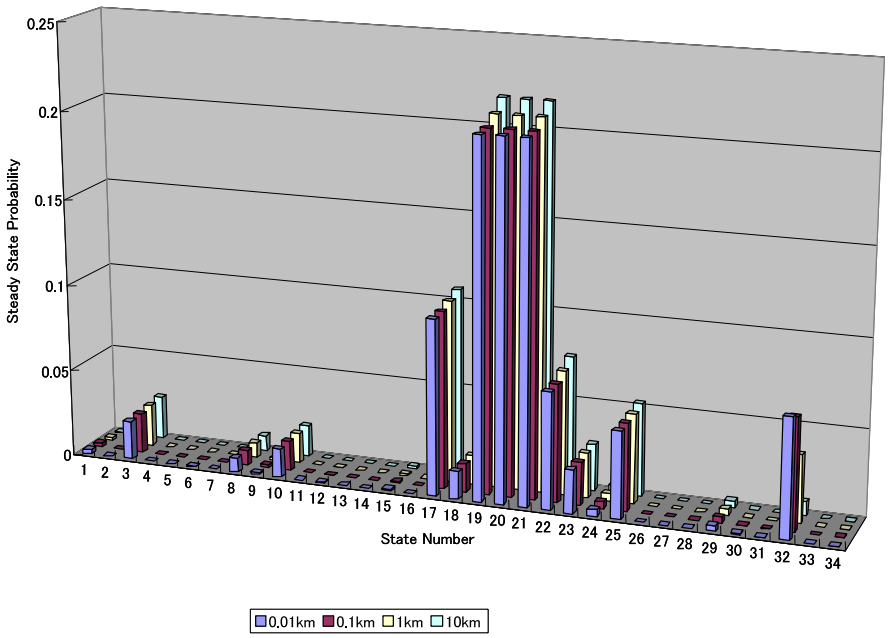
length of ring becomes a million times.

Next, we consider the case of increasing the number of computing nodes (See Fig. 18). Compare the case of $N = 4$ with the case of $N$ is around 10, by increasing the number of node, $P_{19}, P_{20}, P_{21}$ become larger. This is because, when the number of nodes increases, the amount of cache line becomes large and the probability which more then two computing nodes access to the same cache line becomes large. So, steady state probabilities that the node processes the cache coherency protocol become large. However, if the number of nodes become too large, the blocking probability $P_B$ becomes large and influences on other steady state probabilities. Therefore, the case $N \geq 30$, steady state probabilities processing the cache coherency protocol become small as the blocking probability becomes large.

Then we explain about the influence of the change of $s$ (See Fig. 19). While $s$ is large value around $10^{-1}$ or $10^{-2}$, steady state probabilities that are the state of processing cache coherency protocol, are large. However, as the value of $s$ becomes smaller, these probabilities become smaller and $P_1$ and $P_{32}$, which are the probabilities that these state compute and access to the local memory, become larger and total of these probabilities reaches about one and steady state probabilities of the cache coherency protocol processing become small. Consequently, we consider that $s$ can have big impact on the performance of shared memory architecture.

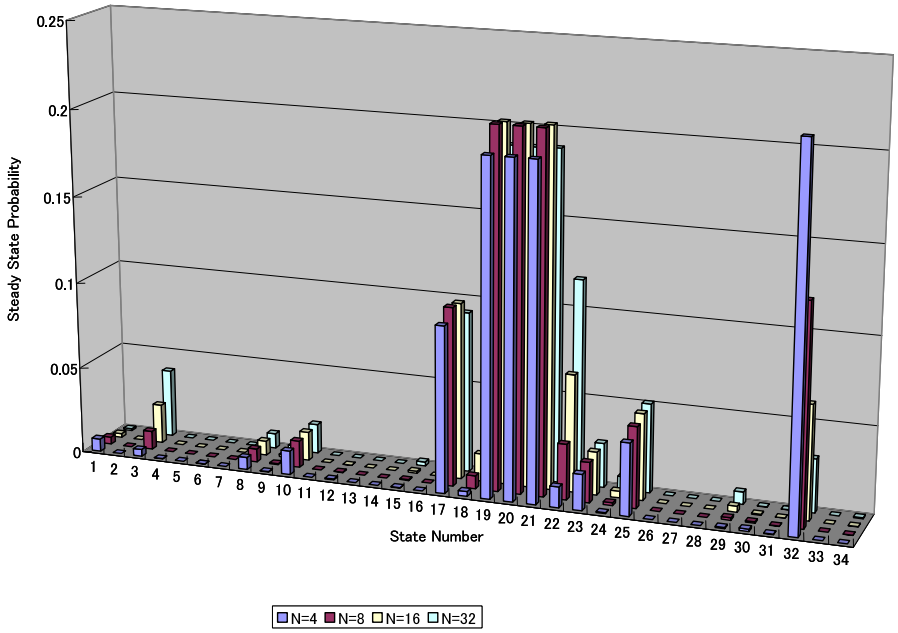



(a) Scenario 1

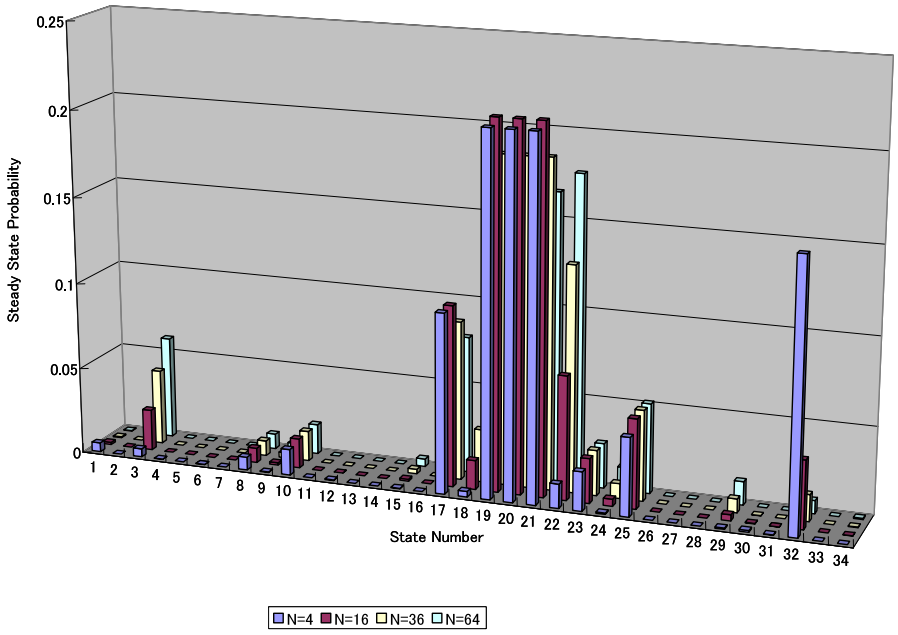



(b) Scenario 2

Figure 17: Distribution of steady state probability of Ring-UMA architecture ($N = 16, s = 10^{-2}$)

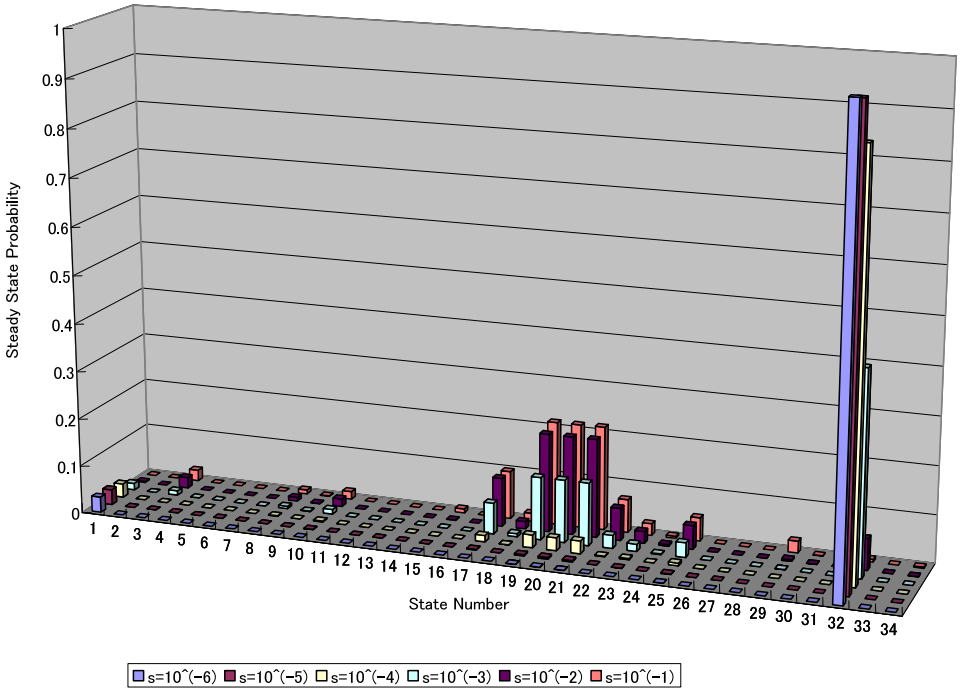



(a) Scenario 1

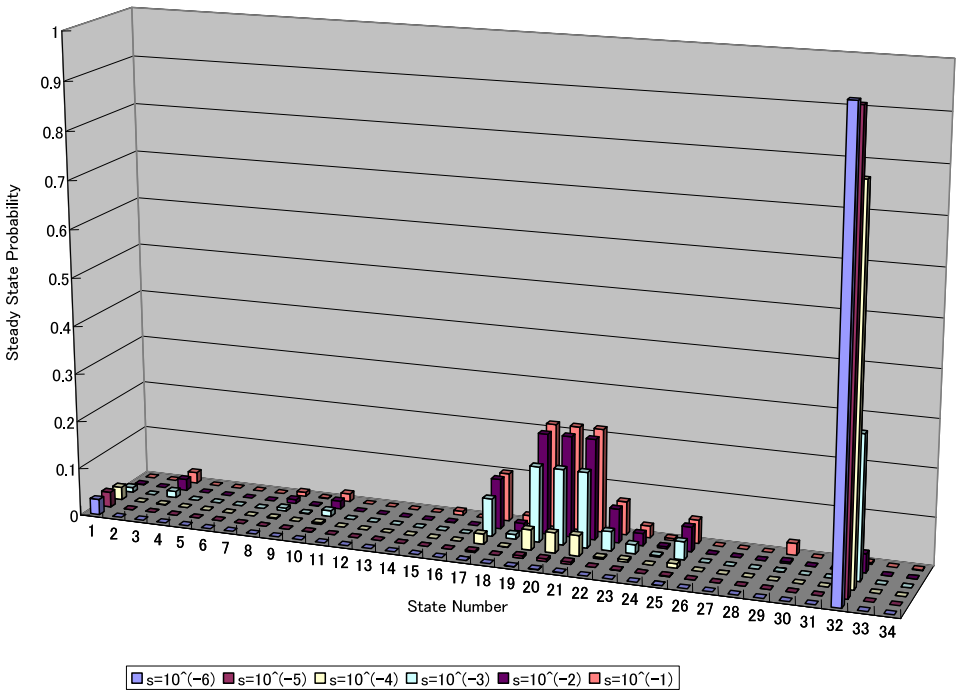



(b) Scenario 2

Figure 18: Distribution of steady state probability of Ring-UMA architecture ($s = 10^{-2}$, $L$ =1km)




(a) Scenario 1



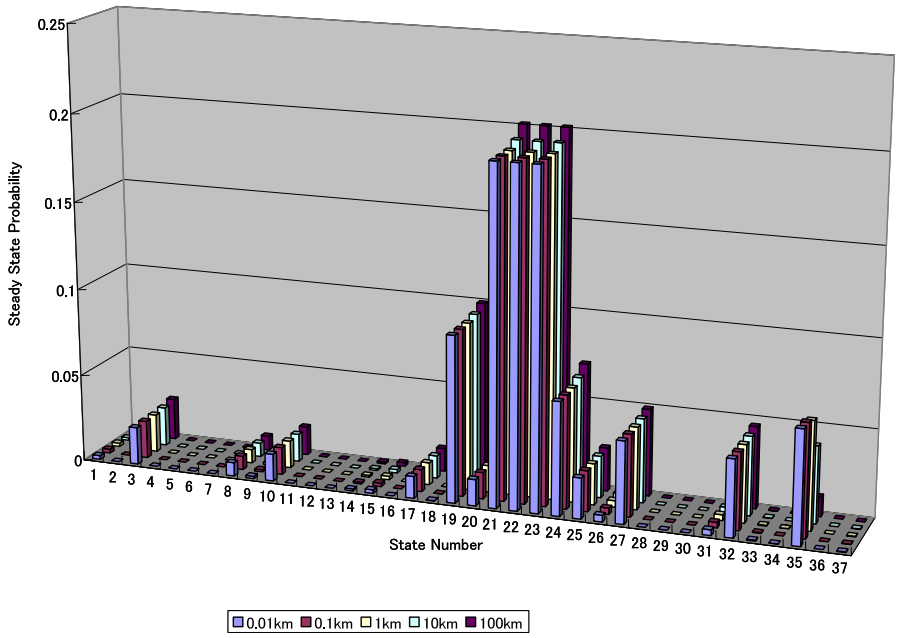

(b) Scenario 2

Figure 19: Distribution of steady state probability of Ring-UMA architecture ($N = 16$, $L =$1 km)

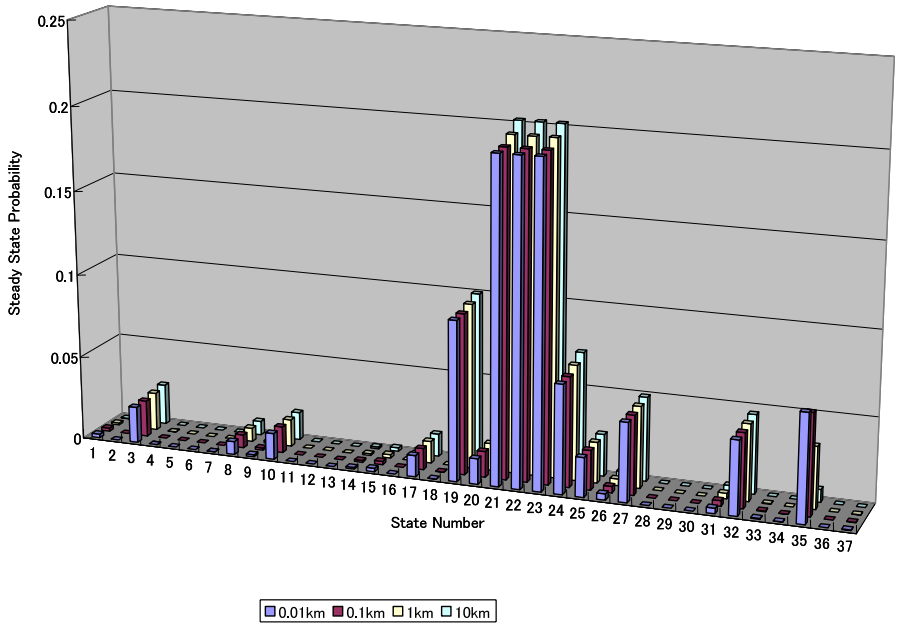
In Ring-NUMA architecture, we can also find these trend described above. Because, there is few difference between Ring-UMA architecture and Ring-NUMA architecture. One important difference is that when cache miss occurred, a computing node may send the copy message. However, the probability that cache miss occurs is low by probability $1 - h$. Therefore, this difference have little influence on the performance of Ring-NUMA architecture (See Fig. 20–Fig. 22).

Fig. 23 through Fig. 25 show the distributions of the steady state probabilities of Mesh-NUMA architecture. In Fig. 24(a), $P_5, P_6, P_7, P_8$, which are the states of processing the cache coherency protocol, are large. These states transit when computing nodes have a cache line in $C$ state and write access to the relative cache line occurs. It takes time that is decided by propagation delay and a request processing time at each computing node to perform the cache coherency. Therefore, residence times of these states are large and as a result, steady probabilities of these become large. On the contrary, in Fig. 23(b), when the distance between computing nodes is 0.01 km and 0.1 km, $P_{21}$ is the largest. This is because, the propagation delay is very small. In addition, the number of nodes passed by messages is lower than that of scenario 1. So, the total request processing time also become lower. As a result, $P_{21}$ becomes large. However, as the distance between computing nodes becomes longer, the propagation delay becomes larger and this reduces the value of $P_{21}$, and steady state probabilities of the cache coherency protocol processing become large.

When, the number of nodes increases, steady state probabilities for the cache coherency protocol processing ($P_5, P_6, P_7, P_{11}$) become large. Because, as the number of nodes increase, the total number of $C$ state cache line and write access to cache lines are increase. As a result, for keeping cache coherency, steady state probabilities for the cache coherency protocol processing become larger and $P_{21}$ becomes small. By keeping the value of $s$ in small, $P_{21}$ can be kept to the probability about 0.9.

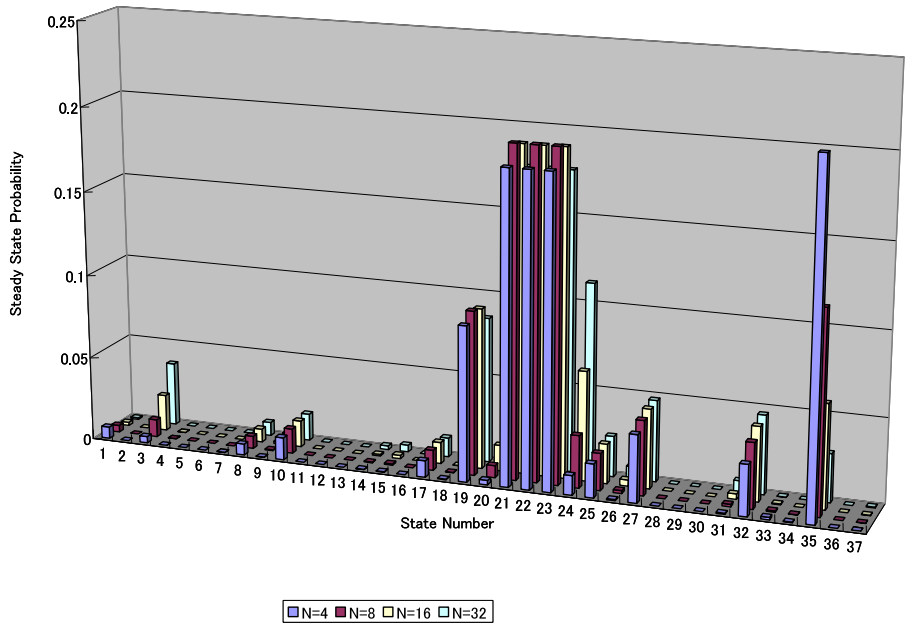



(a) Scenario 1

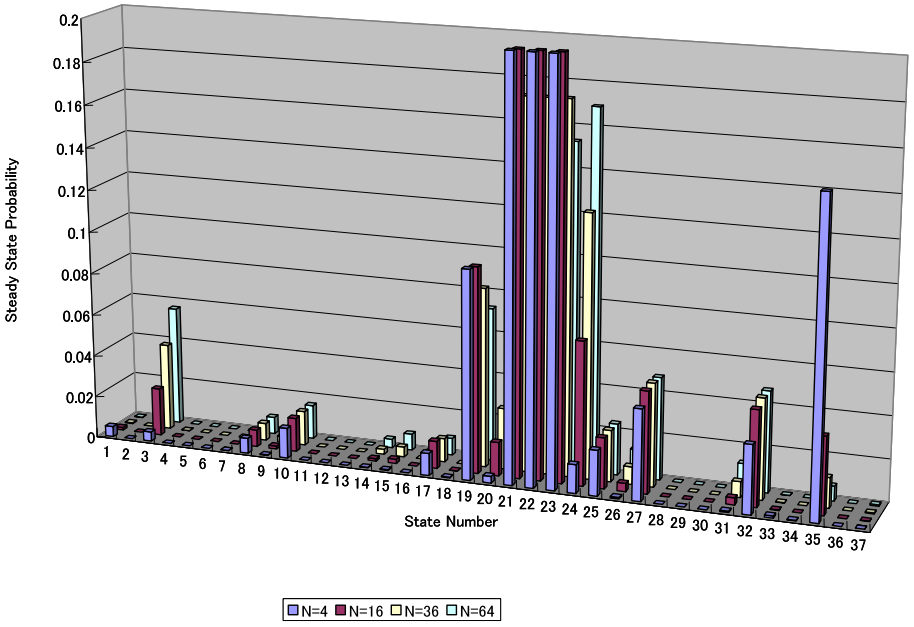



(b) Scenario 2

Figure 20: Distribution of steady state probability of Ring-NUMA architecture ($N = 16, s = 10^{-2}$)

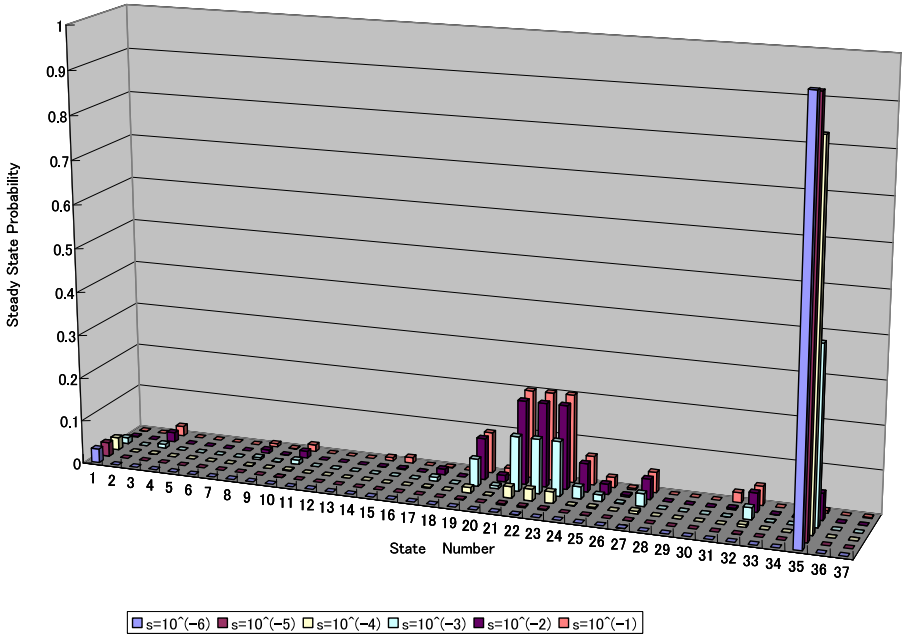



(a) Scenario 1

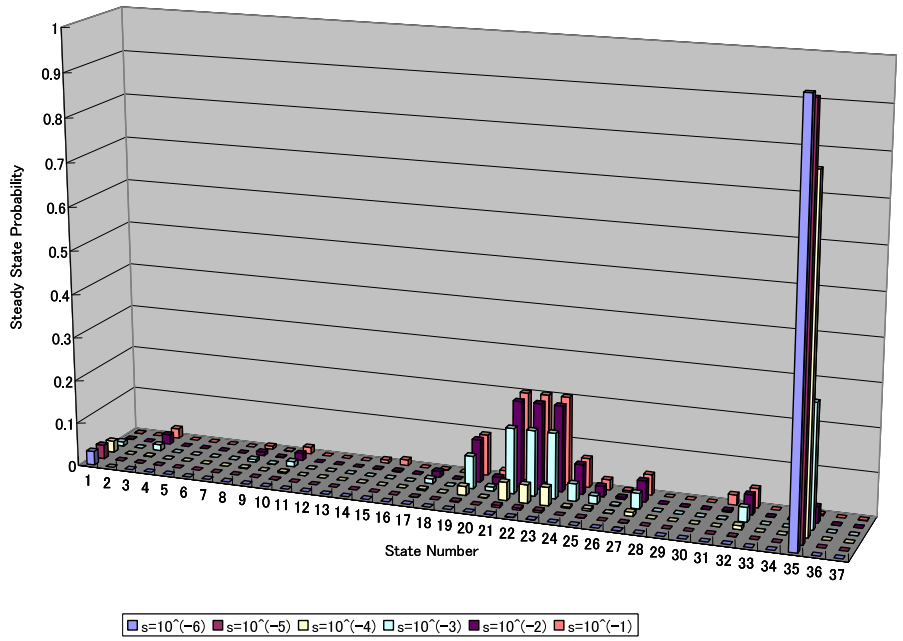



(b) Scenario 2

Figure 21: Distribution of steady state probability of Ring-NUMA architecture ($s = 10^{-2}$, $L$ =1 km)

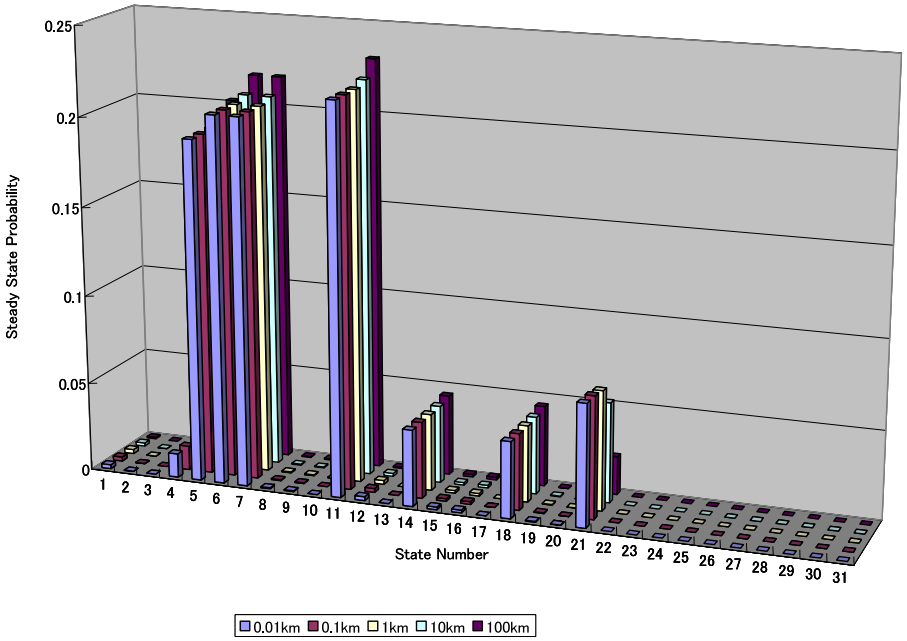



(a) Scenario 1

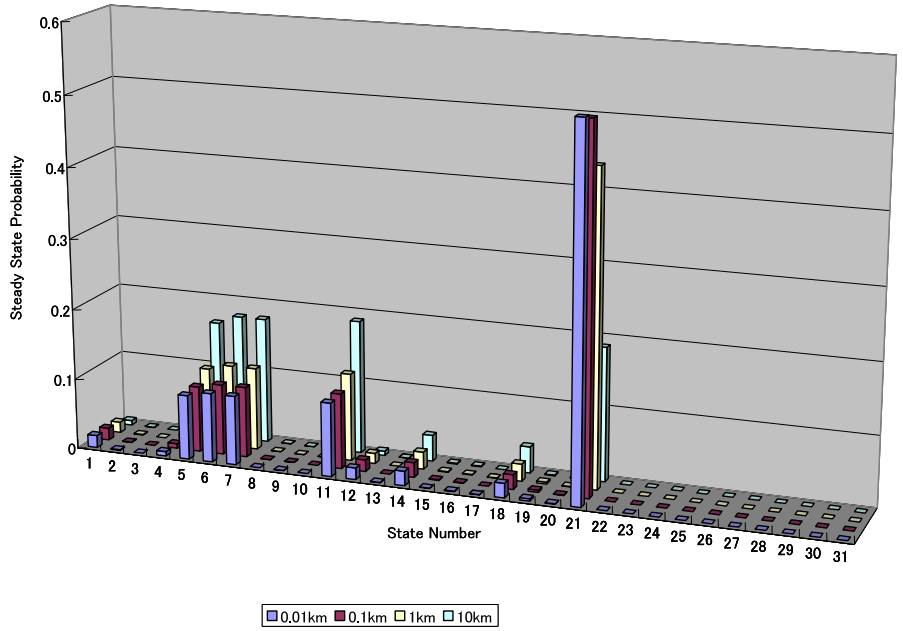



(b) Scenario 2

Figure 22: Distribution of steady state probability of Ring-NUMA architecture ($N = 16$, $L = 1$ km)

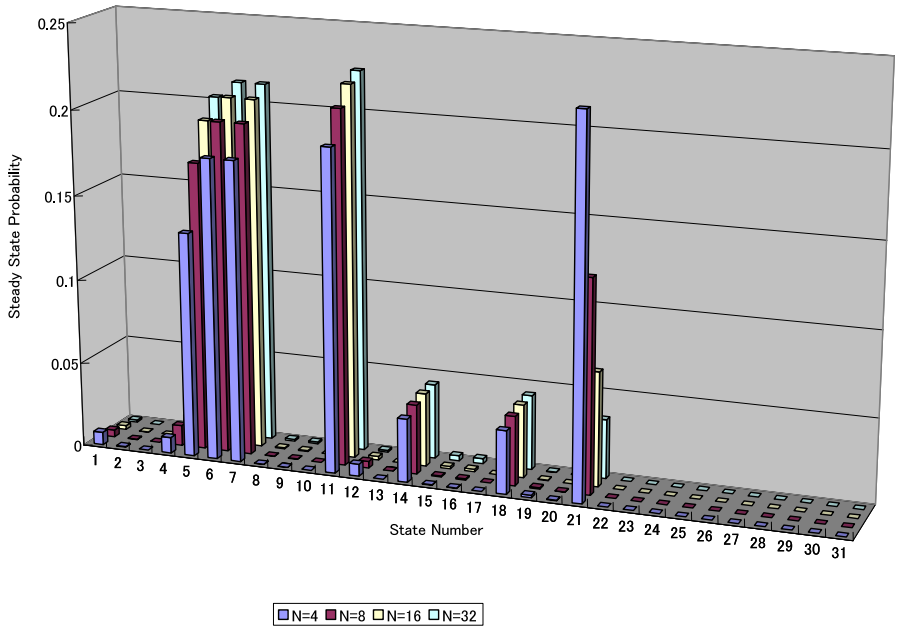



(a) Scenario 1

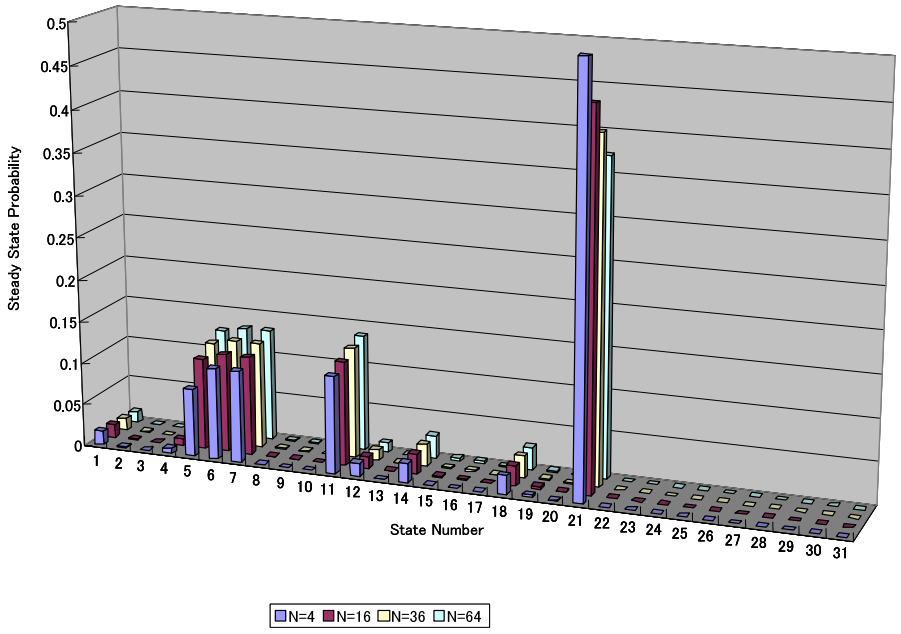



(b) Scenario 2

Figure 23: Distribution of steady state probability of Mesh-NUMA architecture ($N = 16, s = 10^{-2}$)

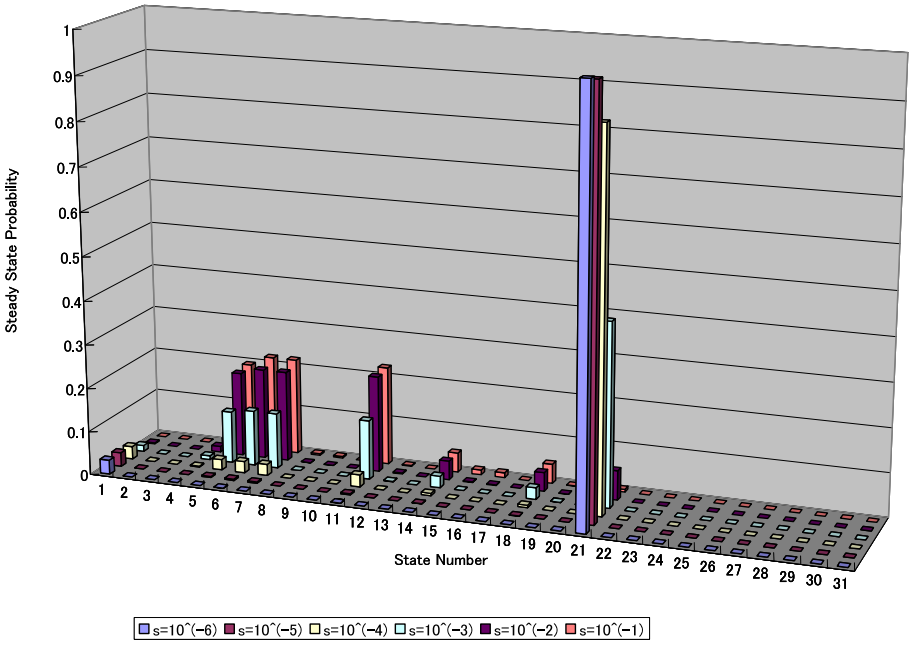



(a) Scenario 1

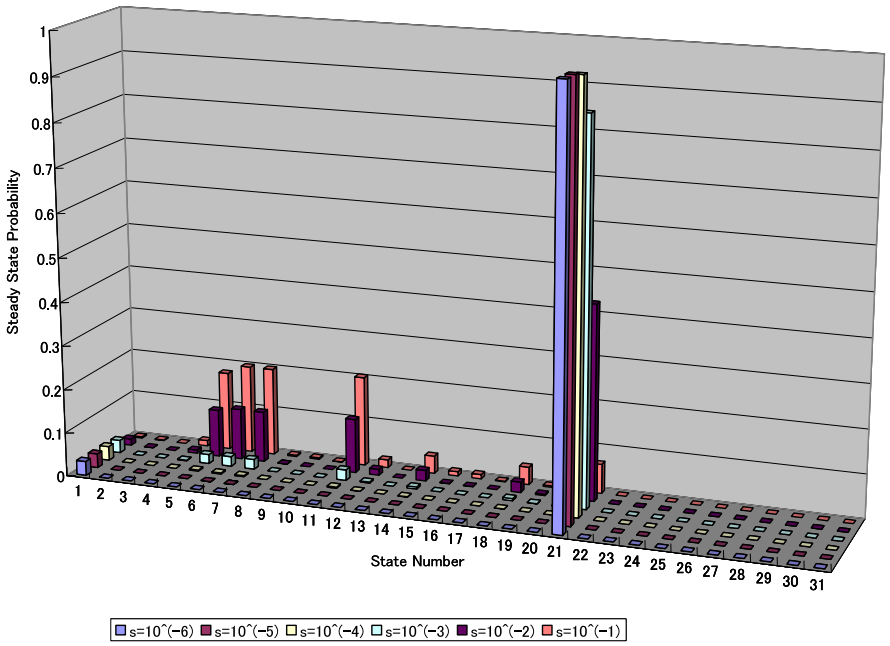



(b) Scenario 2

Figure 24: Distribution of steady state probability of Mesh-NUMA architecture ($s = 10^{-2}$, $L$ =1 km)




(a) Scenario 1




(b) Scenario 2

Figure 25: Distribution of steady state probability of Mesh-NUMA architecture ($N = 16$, $L$ =1 km)

## 5 Evaluation

In this section, we evaluate each shared memory architecture by using the result of numerical analysis obtained in section 4. Here we evaluate three performance measures, that is, network utilization, average memory access time, and computation throughput.

### 5.1 Network utilization

We define the network utilization by following expression.

$$\text{Network utilization} = N \times \sum_{v \in V} (\frac{D_v}{B \times \eta_v} P_v) \tag{23}$$

where $V$ is the set of states where some data are sent, and $D_v$ is size of data transmitted at the state $v$. From this expression, we find that a residence time of each state can influence on network utilization. That is, if a residence time of a state is high, network utilization becomes low. In this study, data is the message (32 byte) or the cache line (4 KByte). We show $V$ and $D_v$ in each architecture below.

- Ring-UMA architecture

  $V = \{10, 15, 19, 20, 21, 25, 29, 34\}$

  $D_{10} = D_{15} = D_{19} = D_{20} = D_{21} = D_{25} = D_{29} = 32$ bit

  $D_{34} = 32 + 4 \times 8 \times 10^3$ bit

- Ring-NUMA architecture

  $V = \{10, 14, 15, 17, 21, 22, 23, 27, 31, 32, 37\}$

  $D_{10} = D_{14} = D_{21} = D_{22} = D_{23} = D_{27} = D_{31} = 32$ bit

  $D_{15} = D_{17} = D_{32} = D_{37} = 32 + 4 \times 8 \times 10^3$ bit

- Mesh-NUMA architecture

  $V = \{4, 14, 23, 24, 26, 31\}$

  $D_4 = D_{14} = D_{23} = D_{26} = 32$ bit

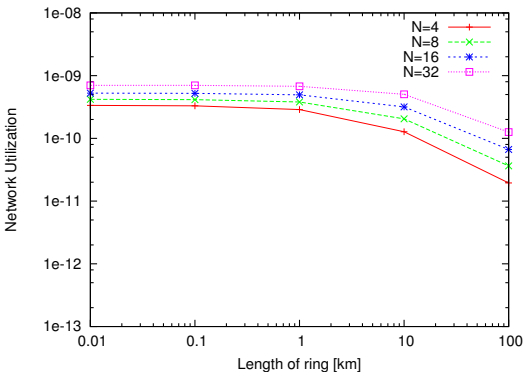
  $D_{24} = D_{31} = 32 + 4 \times 8 \times 10^3$ bit

We find that network utilization is extremely low and less than $10^{-9}$ in each scenario, in all architecture. This is because, computing nodes use network when the processing of cache

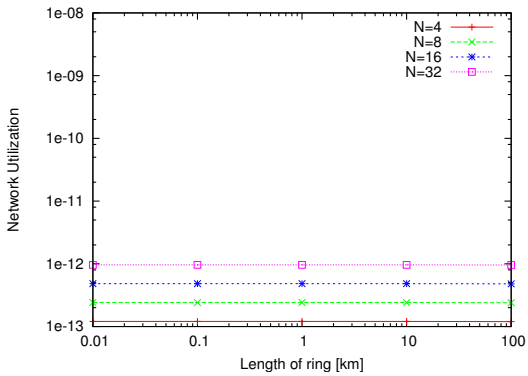
coherency protocol is needed. However, the cache coherency protocol is bursty performed because cache hit ratio is high and most of accesses to the shared memory are read access. Thus, an access to the shared memory end up to be read hit and the read hit access does not require processing of the cache coherency protocol. Moreover, as the length of ring or distance between computing nodes become longer, network utilization becomes low because the propagation delay becomes large.

Compare with case of Ring-UMA architecture and Ring-NUMA architecture in both scenario, network utilization of Ring-UMA architecture is higher than that of Ring-NUMA architecture. The reason is considered as follows. The copy message makes the network utilization relatively small when we think about the entire execution time. That is, the steady state probability of the state where the node sends the copy message makes other steady state probabilities small and this influence on network utilization.

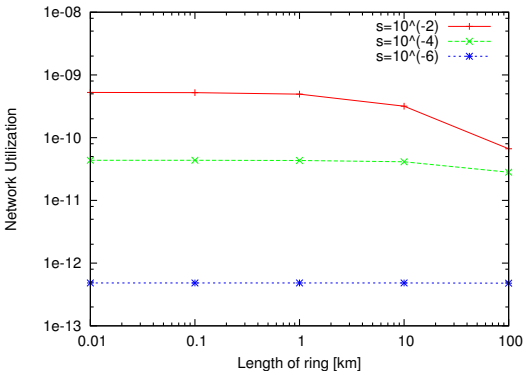
In Mesh-NUMA architecture, network utilization is lower than that of Ring-UMA/NUMA architecture. In Ring-UMA/NUMA architecture, that we designed, transmitted data has to round ring once and source node has to wait the data rounding optical ring. However, in Mesh-NUMA architecture, computing nodes can directly communicate with each other and does not have to wait the message for rounding the optical ring. So average propagation delay of Mesh-NUMA architecture is lower than that of Ring-UMA/NUMA architecture. As a result, the network utilization of Mesh-NUMA architecture becomes higher than Ring-UMA/NUMA architecture. Comparing with the case of scenario 1 and 2, the average propagation delay of scenario 1 is higher than that of scenario 2. So, network utilization of scenario 1 is lower than the case of scenario 2.



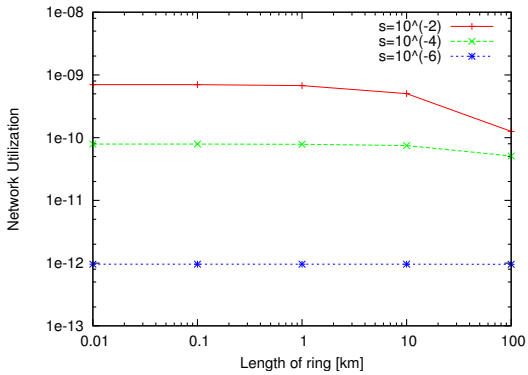

(a) $s = 10^{-2}$




(b) $s = 10^{-6}$

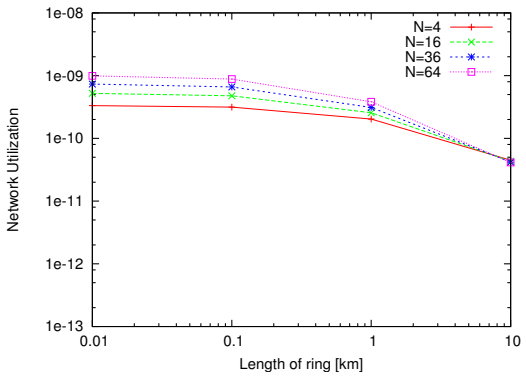



(c) $N = 16$

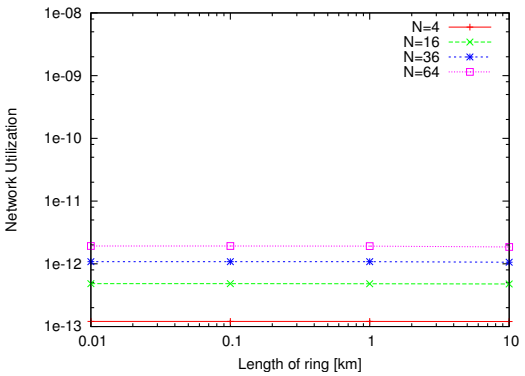



(d) $N = 32$

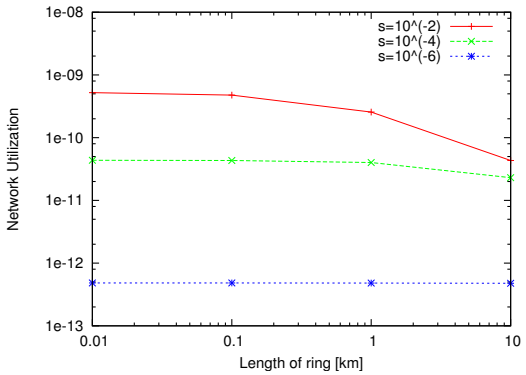
Figure 26: Network utilization of Ring-UMA architecture in scenario 1.



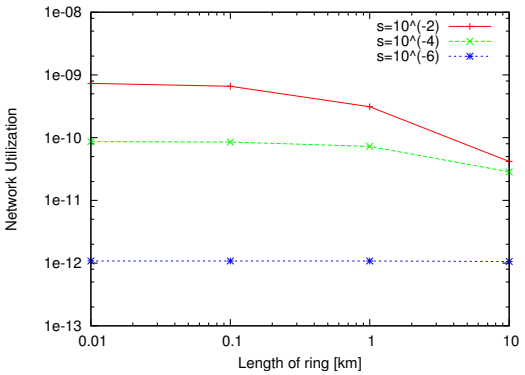

(a) $s = 10^{-2}$




(b) $s = 10^{-6}$

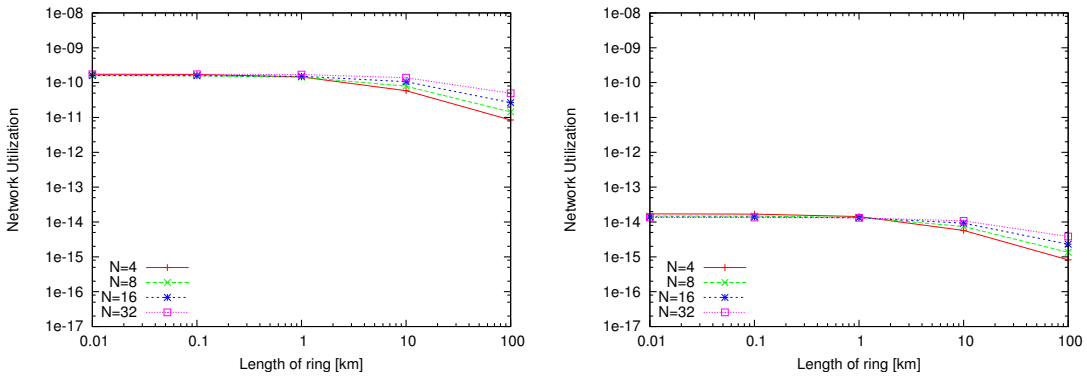



(c) $N = 16$




(d) $N = 32$

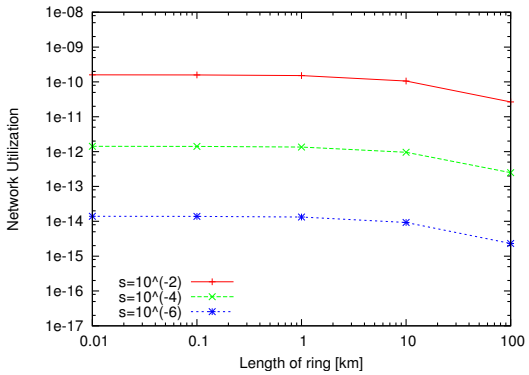
Figure 27: Network utilization of Ring-UMA architecture in scenario 2.



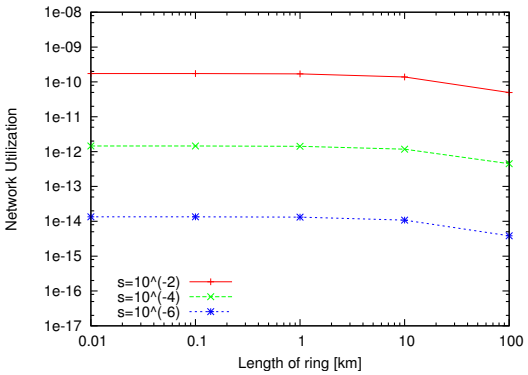

(a) $s = 10^{-2}$




(b) $s = 10^{-6}$

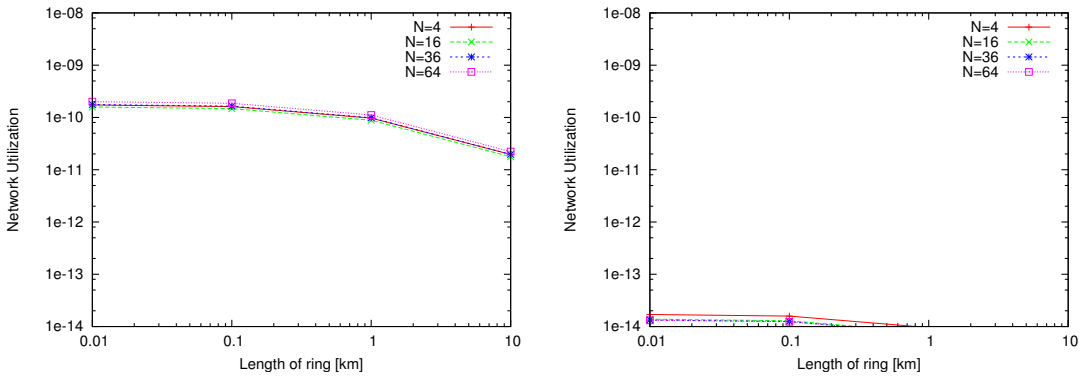



(c) $N = 16$




(d) $N = 32$

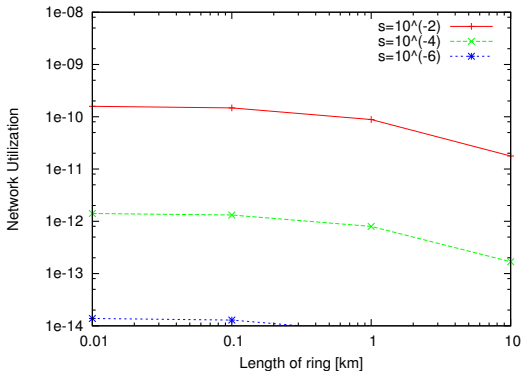
Figure 28: Network utilization of Ring-NUMA architecture in scenario 1.



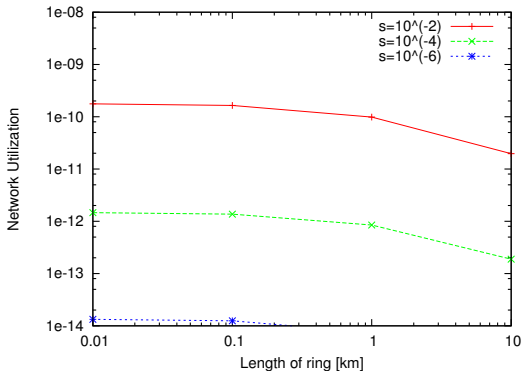

(a) $s = 10^{-2}$




(b) $s = 10^{-6}$

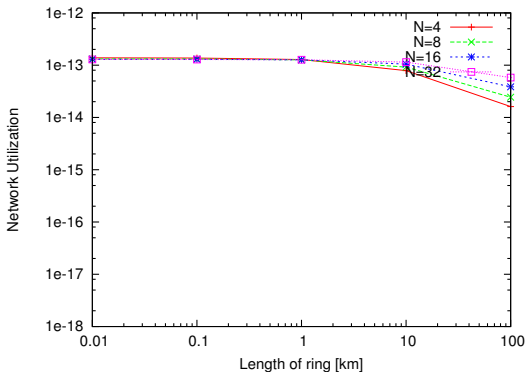



(c) $N = 16$

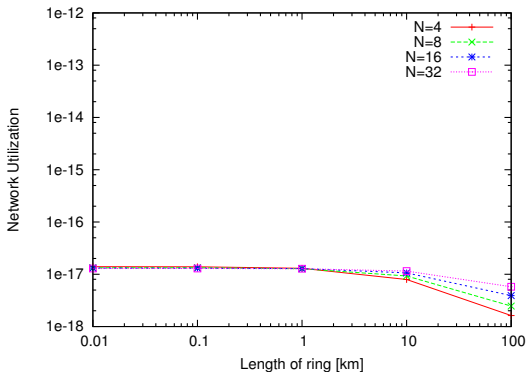



(d) $N = 36$

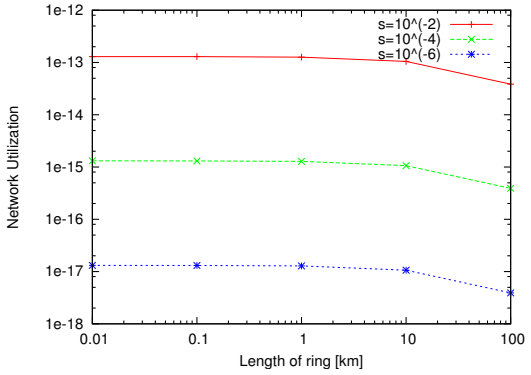
Figure 29: Network utilization of Ring-NUMA architecture in scenario 2.



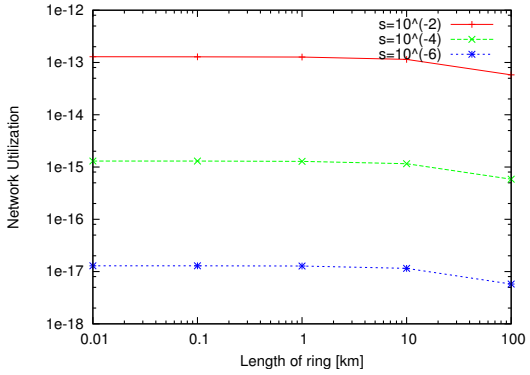

(a) $s = 10^{-2}$




(b) $s = 10^{-6}$

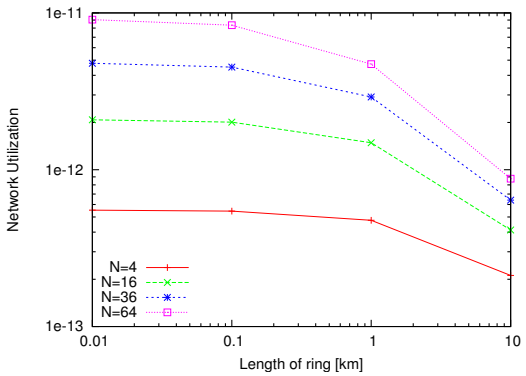



(c) $N = 16$

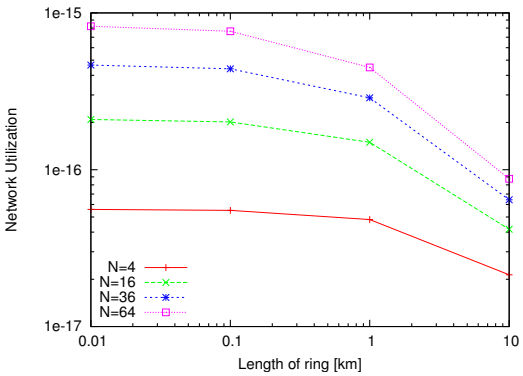



(d) $N = 32$

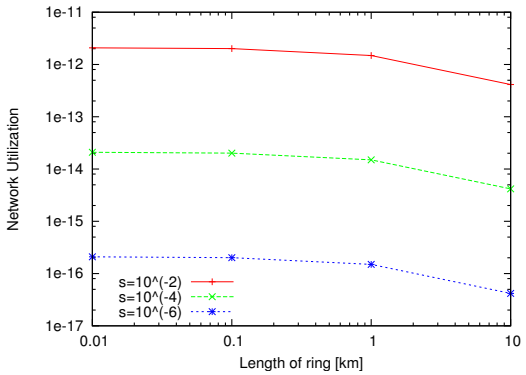
Figure 30: Network utilization of Mesh-NUMA architecture in scenario 1.



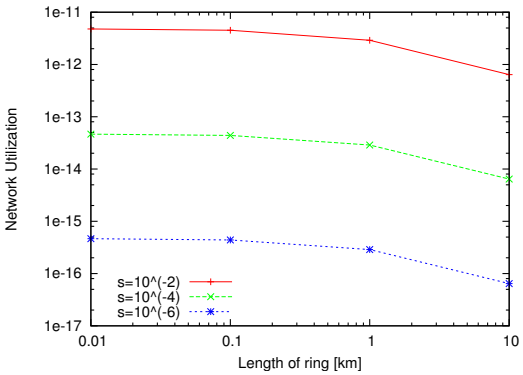

(a) $s = 10^{-2}$




(b) $s = 10^{-6}$




(c) $N = 16$




(d) $N = 36$

Figure 31: Network utilization of Mesh-NUMA architecture in scenario 2.

### 5.2 Average memory access time to the shared memory

We define the average memory access time, denoted as $t_{share}$, to the shared memory as following expression.

$$t_{share} = r \times t_{sr} + w \times t_{sw} \quad (24)$$

$$t_{sr} = h \times \sum_{u \in R_h} \delta_u \eta_u + (1-h) \times \sum_{v \in R_m} \delta_v \eta_v \quad (25)$$

$$t_{sw} = h \times \sum_{u \in W_h} \delta_u \eta_u + (1-h) \times \sum_{v \in W_m} \delta_v \eta_v \quad (26)$$

where $t_{sr}$ is the average memory access time of read access, $t_{sw}$ is the average memory access time of write access. $R_h$ is the set of states which they can transit when the read access is cache hit, $R_m$ is the set of state which they can transit when the read access is cache miss. $W_h$ is the set of states where they can transit when the write access is cache hit, $W_m$ is the set of state which they can transit when the write access is cache miss. $\delta_u$ is the probability, that the state $u$ is passed by the probability $\delta_u$.

We show $t_{sr}$ and $t_{sw}$ of each architecture below.

- Ring-UMA architecture

$$\begin{aligned} t_{sr} &= \eta_2 + P_B\eta_{22} + h\eta_{31} \\ &\quad +(1-h)(\eta_{23} + P_B\eta_{24} + \eta_{25} + P_d(\eta_{26} + \eta_{29}) + P_c\eta_{27} + (1 - P_c - P_d)\eta_{28} + P_x\eta_{30} + \eta_{31}) \\ t_{sw} &= \eta_2 + P_B\eta_3 + h(\eta_4 + w(\eta_5 + \eta_6) + r(\eta_7 + \eta_{17} + B\eta_{18} + \eta_{19} + \eta_{20} + \eta_{21})) \\ &\quad +(1-h)(\eta_8 + P_B\eta_9 + \eta_{10} + (1 - c - d)(\eta_{11} + P_x\eta_{12} + \eta_6) \\ &\qquad + c(\eta_{13} + P_x\eta_{16} + \eta_{19} + \eta_{20} + \eta_{21}) \\ &\qquad + d(\eta_{14} + \eta_{15} + P_x\eta_{16} + \eta_{19} + \eta_{20} + \eta_{21})) \end{aligned}$$

- Ring-NUMA architecture

$$\begin{aligned} t_{sr} &= \eta_2 + P_B\eta_{24} + h\eta_{34} \\ &\quad +(1-h)(\eta_{25} + P_B\eta_{26} + \eta_{27} + P_d(\eta_{28} + \eta_{31}) + P_c\eta_{29} \\ &\qquad + (1 - P_c - P_d)\eta_{30} + \eta_{32+}P_x\eta_{33} + \eta_{34}) \\ t_{sw} &= \eta_2 + P_B\eta_3 + h(\eta_4 + w(\eta_5 + \eta_7) + r(\eta_6 + \eta_{19} + P_B\eta_{20} + \eta_{21} + \eta_{22} + \eta_{23})) \\ &\quad +(1-h)(\eta_8 + P_B\eta_9 + \eta_{10} + (1 - c - d)(\eta_{11} + \eta_{17} + P_x\eta_{18} + \eta_5 + \eta_7) \end{aligned}$$

$$
\begin{aligned}
&+ c(\eta_{12} + \eta_{15} + P_x\eta_{16} + \eta_{21} + \eta_{22} + \eta_{23}) \\
&+ d(\eta_{13} + \eta_{14} + \eta_{15} + P_x\eta_{16} + \eta_{21} + \eta_{22} + \eta_{23}))
\end{aligned}
$$

- Mesh-NUMA architecture

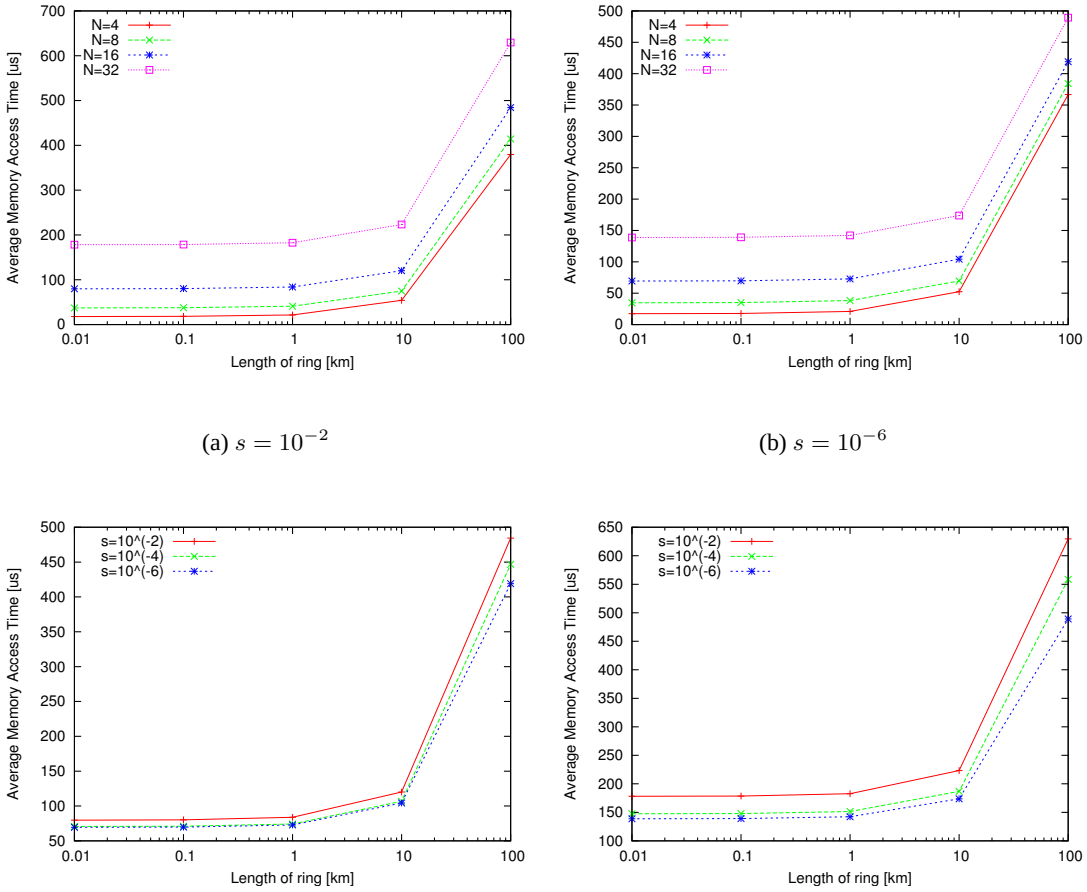
$$
\begin{aligned}
t_{sr} &= h(\eta_2 + \eta_{20}) \\
&\quad +(1-h)P_d(\eta_{14} + \eta_{15} + \eta_{16} + \eta_{17} + \eta_{18} + P_x\eta_{19} + \eta_{20}) \\
&\quad +(1-h)(1-P_d)(\eta_{14} + \eta_{18} + P_x\eta_{19} + \eta_{20}) \\
t_{sw} &= h(\eta_2 + \eta_3 + \eta_{13} + r(\eta_5 + \eta_6 + \eta_7 + \eta_{11} + P_x\eta_{12})) \\
&\quad +(1-h)P_d(\eta_2 + \eta_4 + \eta_8 + \eta_9 + \eta_{10} + \eta_{11} + P_x\eta_{12} + \eta_{13} \\
&\quad +(1-h)P_c(\eta_2 + \eta_4 + \eta_6 + \eta_7 + \eta_{11} + P_x\eta_{12} + \eta_{13} \\
&\quad +(1-h)(1-P_c-P_d)(\eta_2 + \eta_4 + \eta_6 + \eta_7 + \eta_{11} + P_x\eta_{12} + \eta_{13})
\end{aligned}
$$

From these expressions, we can find that average memory access time is determined by residence time of states in model. So, the length of ring (in scenario 1) or the distance between computing nodes (in scenario 2) have big impact on average memory access time because the residence time is decided by them.

Fig. 32–Fig. 37 shows average memory access time of each architecture and each scenario. As mentioned before, as the length of ring or the distance between computing nodes become longer, average memory access time remarkably become larger. However, in Ring-UMA/NUMA architecture, while the length of ring is small (less than 1 km), average memory access time is larger than propagation delay. The reason is frame processing time at the network interface. We set this value at 3 $\mu$s by referring our implemented prototype system. This value is large and while the length of ring is short, total frame processing delay at each computing nodes occupies the most of average memory access time.

As the number of increase, the number of nodes passed by data rounding optical ring become large in scenario 2. So, in scenario 2, the case of $N = 64$, the length of ring reaches 640 km and this is very long. Therefore, Ring-UMA/NUMA architecture in scenario 2 takes time to access to the shred memory longer than that of in scenario 1.

On the contrary, in Mesh-NUMA architecture (Fig. 36, Fig. 37), average memory access time is smaller than that of Ring-UMA/NUMA architecture. This is because a CPU does not have to




(a) $s = 10^{-2}$

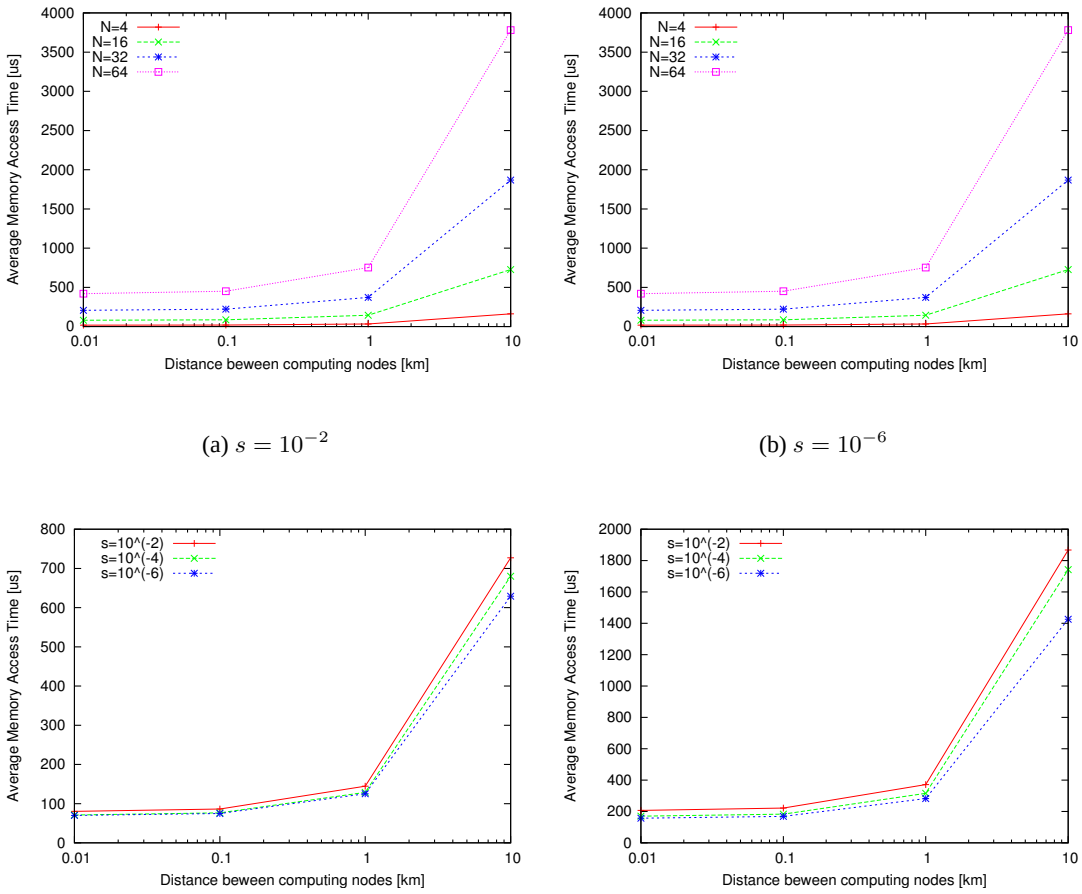
(b) $s = 10^{-6}$

(c) $N = 16$

(d) $N = 32$

Figure 32: Average memory access time of Ring-UMA architecture in scenario 1.

wait the data rounding optical ring, that is broadcasting, and can communicate each other directly. So the propagation delay becomes smaller than that of Ring-UMA/NUMA architecture and average memory access time becomes small. Especially in scenario 2, the physical topology is grid topology. So, average propagation delay become lower than that of scenario 1 that organize ring topology in physical. Therefore, average memory access time in scenario 2 is lower than that of scenario 1.



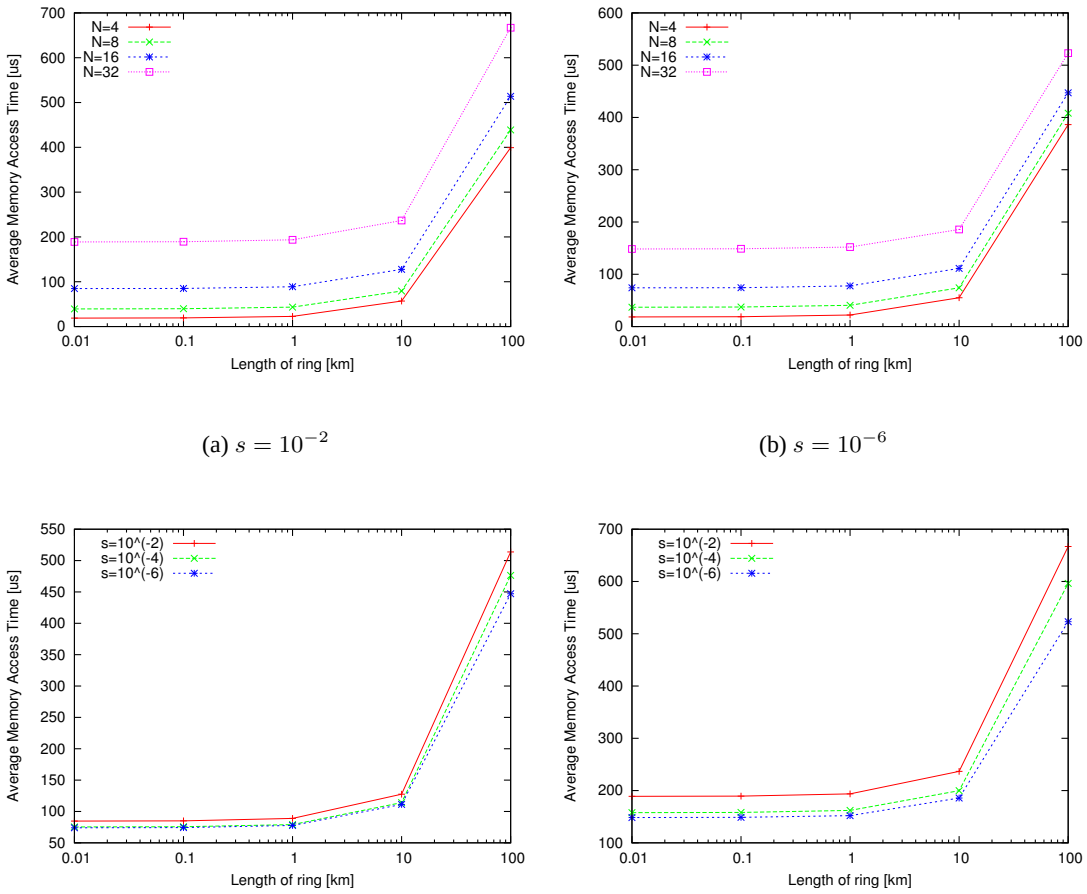

(a) $s = 10^{-2}$

(b) $s = 10^{-6}$

(c) $N = 16$

(d) $N = 36$

Figure 33: Average memory access time of Ring-UMA architecture in scenario 2.



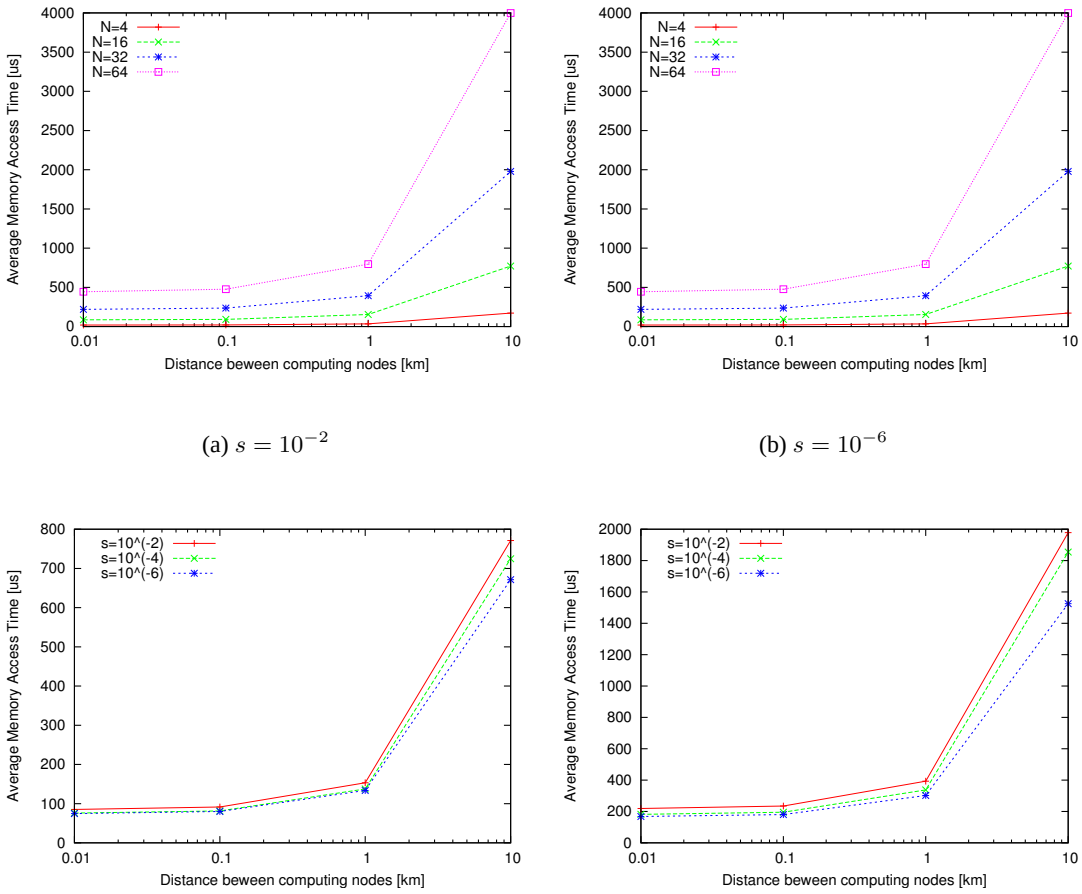

(a) $s = 10^{-2}$

(b) $s = 10^{-6}$

(c) $N = 16$

(d) $N = 32$

Figure 34: Average memory access time of Ring-NUMA architecture in scenario 1.



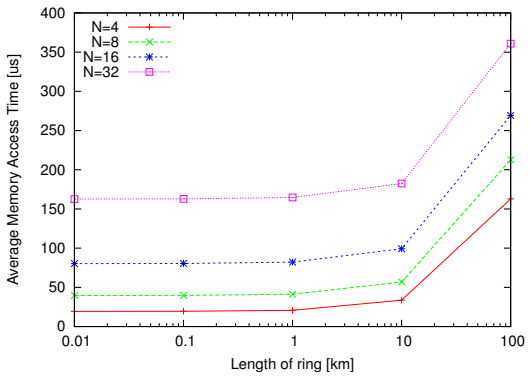

(a) $s = 10^{-2}$

(b) $s = 10^{-6}$

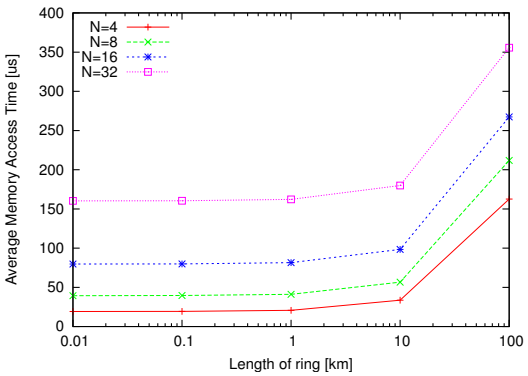
(c) $N = 16$

(d) $N = 36$

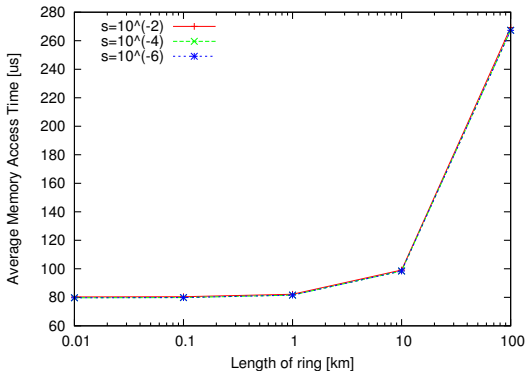
Figure 35: Average memory access time of Ring-NUMA architecture in scenario 2.



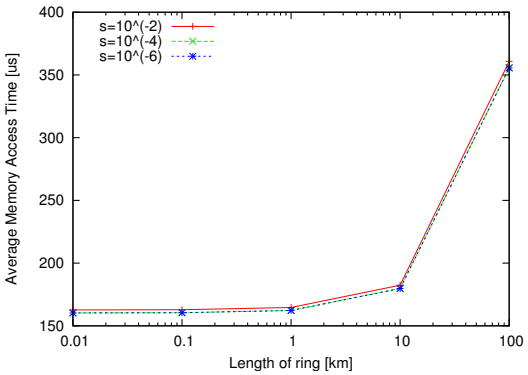

(a) $s = 10^{-2}$




(b) $s = 10^{-6}$

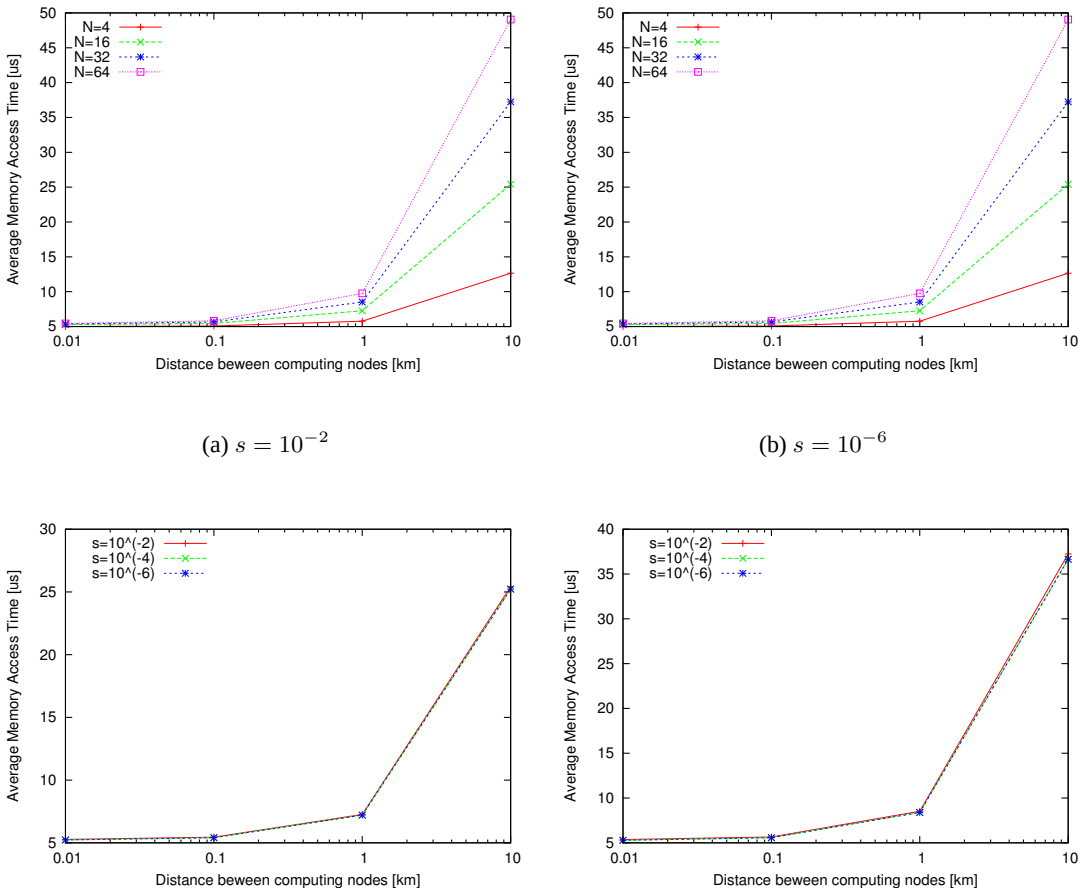



(c) $N = 16$




(d) $N = 32$

Figure 36: Average memory access time of Mesh-NUMA architecture in scenario 1.




(a) $s = 10^{-2}$

(b) $s = 10^{-6}$

(c) $N = 16$

(d) $N = 36$

Figure 37: Average memory access time of Mesh-NUMA architecture in scenario 2.

### 5.3 Computation throughput

We define the computation throughput by following expression. In this study, we use the computation throughput as MIPS (Million Instruction Per Second).

$$\text{MIPS} = \frac{N}{0.8 \times \eta_1 + 0.2 \times s \times t_{share} + 0.2 \times (1-s) \times t_{private}} \tag{27}$$

where $t_{private}$ is the average memory access time to the local memory. We obtain $t_{private}$ as follows.

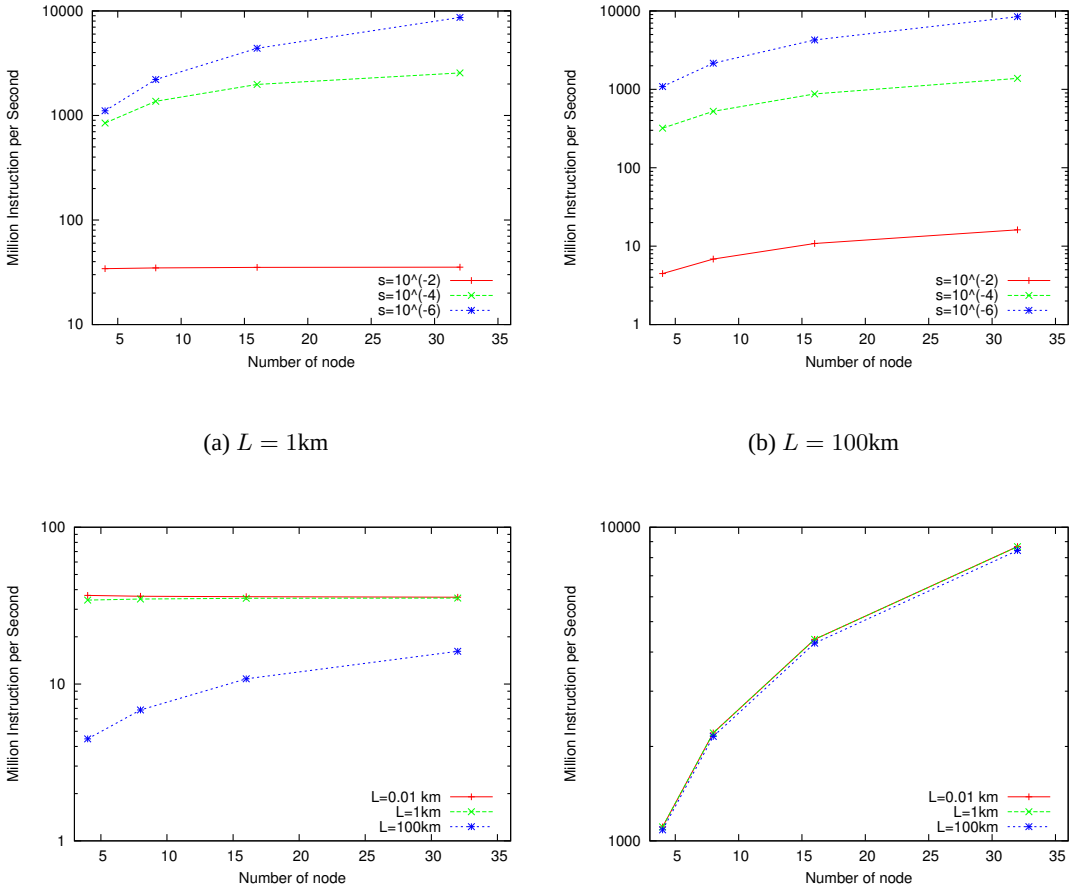
$$t_{private} = h \times t_1 + (1-h) \times (t_1 + t_2) \tag{28}$$

Fig. 38–Fig. 43 show the computation throughput of each architecture. We assume that the frequency of a CPU is 2 GHz. So, if a CPI (Clock Per Instruction) of this CPU is 10, this CPU achieves 200 MIPS. Therefore, we set our first goal to achieve over 200 MIPS. Second goal is to achieve 1000 MIPS.

In Ring-UMA architecture, as the value of $s$ become lower, we can achieve high throughput, on the contrary, while $s$ is large, we can not achieve our goal. The case of $s$ is large, the cache coherency protocol is performed and calculation is stalled. Moreover, the things that $s$ has large value is that a CPU accesses to the shared memory many time. This means that data is frequently rounding optical ring and it takes time to propagation delay. Therefore, while $s$ is large, the length of ring can also influence on the computation throughput badly. On the contrary, while $s$ is low, the length of ring influence on the computation throughput so much and we can achieve our second goal. In scenario 1 and the case of $s = 10^{-6}$, the difference between the case of $L = 100$ km and $L =$1 km are less than 1 %. These facts described above can be found in scenario 2.

Ring-NUMA architecture has similar trend to Ring-UMA architecture in both scenario. In Ring-NUMA architecture, the computation throughput decreases slightly compare to the Ring-UMA architecture. This is because the accesses to the home memory can be processed when cache miss occurs. However, the access to the home node rarely occurs because cache hit ratio is high. Therefore, the computation throughput is not remarkably decreased.

We can found that Mesh-NUMA architecture achieves the high computation throughput and this is about to reach 10000 MIPS. In scenario 1, the trend is similar to the case of Ring-UMA/NUMA architecture of scenario 1. This is because they organize ring topology in physical. However, Mesh-NUMA architecture organize mesh topology in logical, the average propagation delay is




(a) $L = 1$km

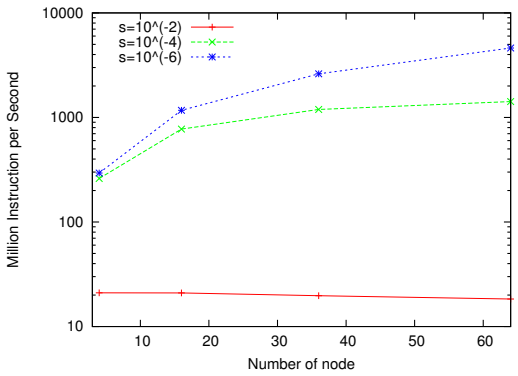
(b) $L = 100$km

(c) $s = 10^{-2}$

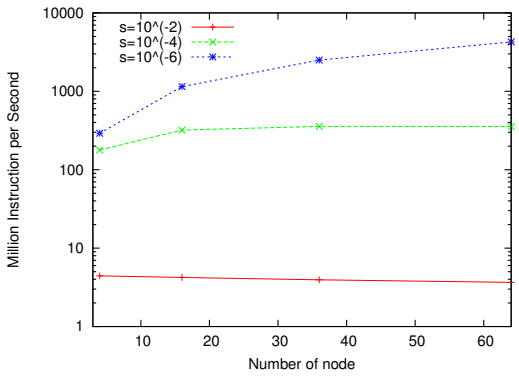
(d) $s = 10^{-6}$

Figure 38: Computation throughput of Ring-UMA architecture in scenario 1.

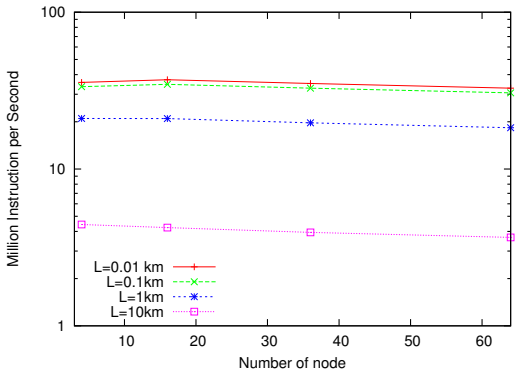
lower than that of Ring-UMA/NUMA architecture. Therefore, we can achieve high computation throughput, which is about to over 10000 MIPS, than that of Ring-UMA/NUMA architecture. In scenario 2, we can achieve higher throughput than that of Ring-UMA/NUMA architecture of scenario 2. The reason is the average propagation delay is small compare to the case of Ring-UMA/NUMA architecture of scenario 2. They can establish the shortest wavelength path among computing nodes. Therefore, in spite of $s$ is large, the influence of the distance between computing nodes is low and we can get higher computation throughput than that of Ring-UMA/NUMA architecture. Especially the case of $s = 10^{-6}$, the computation throughput is beyond 10000 MIPS.



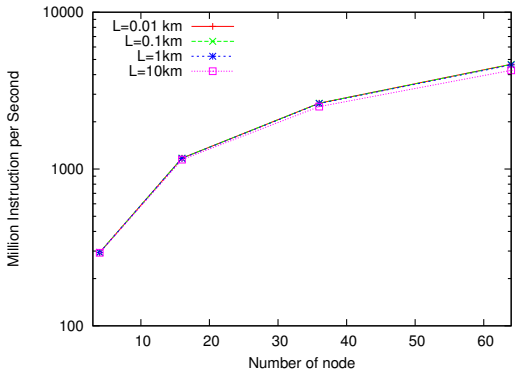

(a) $L = 1$km




(b) $L = 10$km

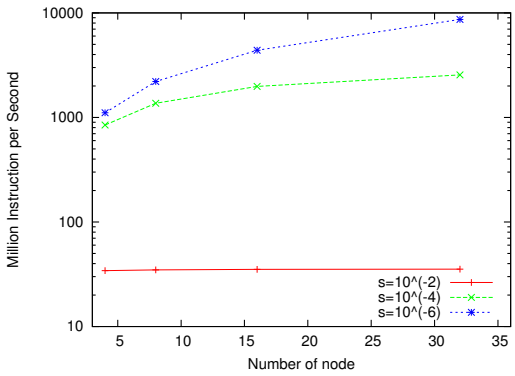



(c) $s = 10^{-2}$

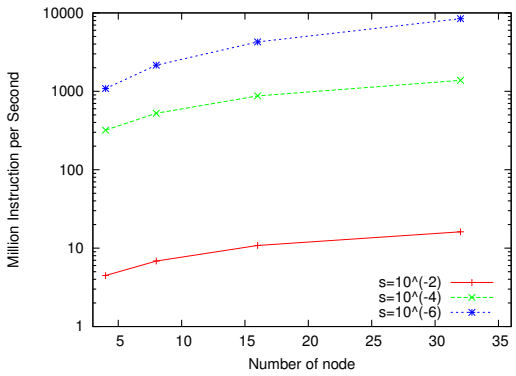



(d) $s = 10^{-6}$

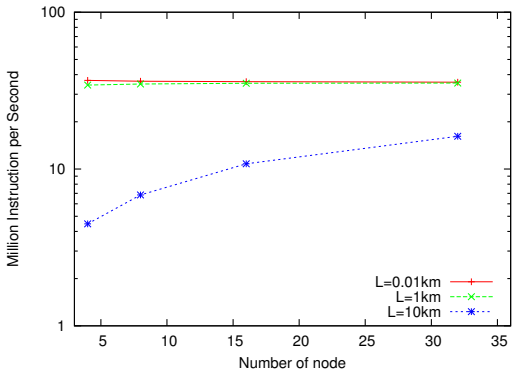
Figure 39: Computation throughput of Ring-UMA architecture in scenario 2.



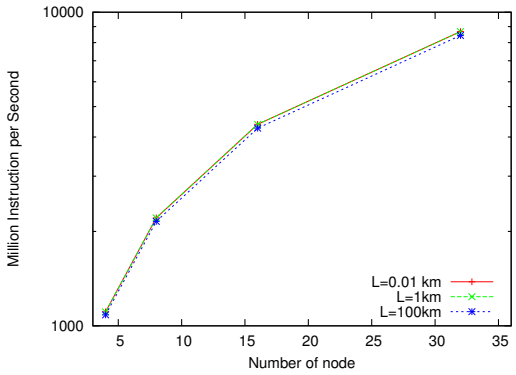

(a) $L = 1$km




(b) $L = 100$km

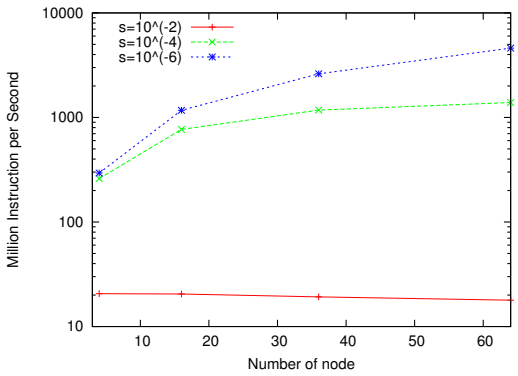



(c) $s = 10^{-2}$

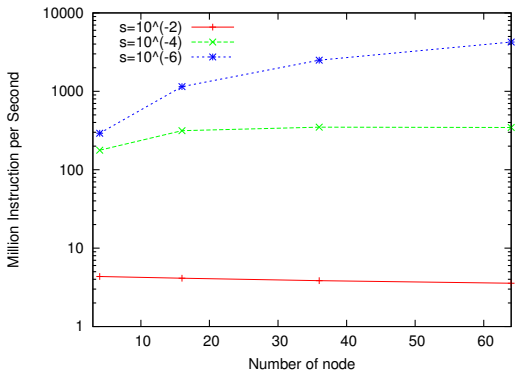



(d) $s = 10^{-6}$

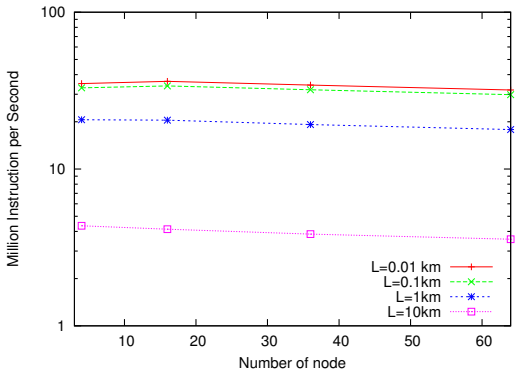
Figure 40: Computation throughput of Ring-NUMA architecture in scenario 1.



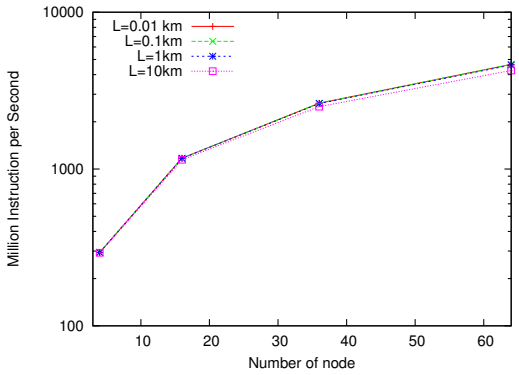

(a) $L = 1$km




(b) $L = 10$km

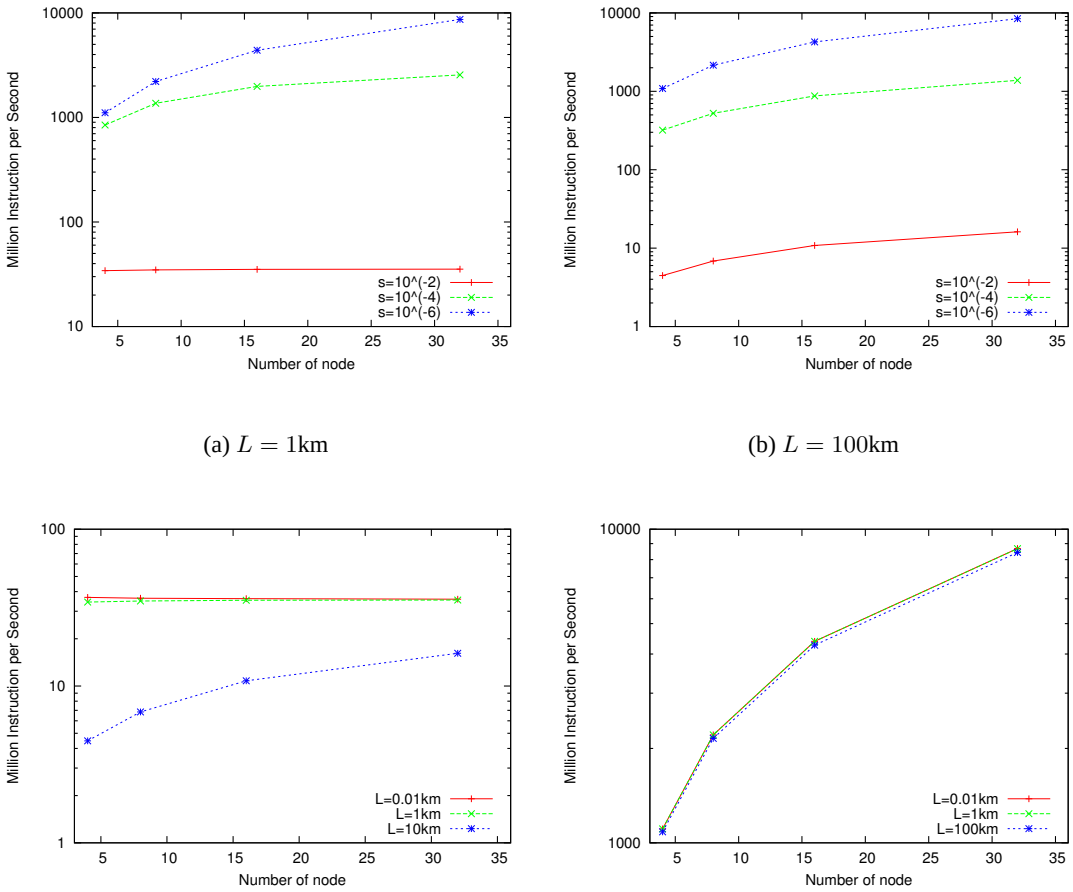



(c) $s = 10^{-2}$




(d) $s = 10^{-6}$

Figure 41: Computation throughput of Ring-NUMA architecture in scenario 2.



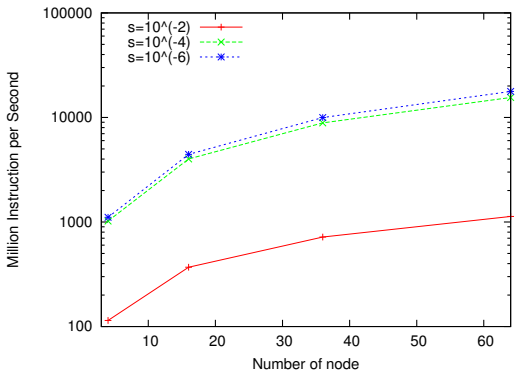

(a) $L = 1\text{km}$

(b) $L = 100\text{km}$

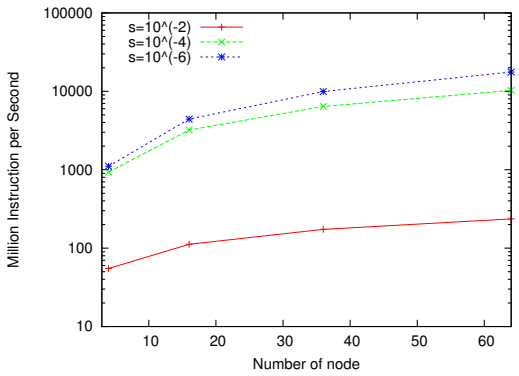
(c) $s = 10^{-2}$

(d) $s = 10^{-6}$

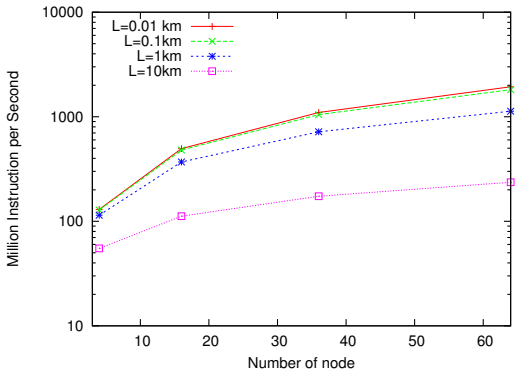
Figure 42: Computation throughput of Mesh-NUMA architecture in scenario 1.



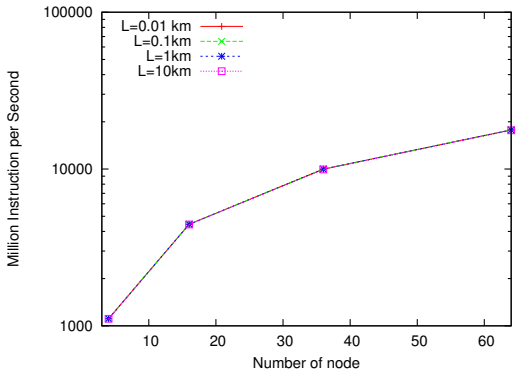

(a) $L = 1$km




(b) $L = 10$km




(c) $s = 10^{-2}$




(d) $s = 10^{-6}$

Figure 43: Computation throughput of Mesh-NUMA architecture in scenario 2.

# 6 Conclusion

In this thesis, we designed and evaluated some shared memory architectures in $\lambda$ computing environment that we proposed as new distributed computing environment. We confirm that shared memory architecture in $\lambda$ computing environment have good performance.

We designed three architecture; Ring-UMA architecture, Ring-NUMA architecture and Mesh-NUMA architecture in section 3. We investigated the characteristic factors that influence on the performance of shared memory architecture and picked up four factors; topology, memory access model, cache coherency protocol, and realization method for cache coherency protocol. By taking these factors into account, we designed control messages and behavior of the shared memory architecture.

Then we modeled and analyzed shared memory architectures we designed. We utilized semi-Markov process for modeling and analyzing. By modeling these architecture, we made state transition diagrams of each architecture. And by using these state transition diagrams, we analyzed and obtained the distribution of the steady state probability.

At last, by utilizing steady state probabilities, we evaluated shared memory architectures we designed. As a result, we found that network utilization was very low in all architecture. So, congestion or packet loss which often happens on the Internet does not occur. Therefore, we confirm that we can provide high reliability communication line to computing nodes. On the other hand, the average memory access times is relatively large and we found that this may be influenced on the network. However, we can achieve high computation throughput in each architecture by keeping the ratio of access to the shared memory low. Especially, we can achieve high computation throughput in Mesh-NUMA architecture.

While Mesh-NUMA architecture can achieve the high computation throughput, we do not take into account about the number of wavelength for designing. That is, we assume that we can freely use wavelength path with no restriction. However, the wavelength is actually limited resources and the number of wavelength which each computing node can use is restricted. So, we have to investigate a new architecture by taking into account this. In addition, we do not consider about the synchronization among computing nodes. However, synchronization among computing nodes is very important. So, a new synchronization method suitable for $\lambda$ computing is required. Moreover, keeping the ratio of access to the shared memory is realizable but it requires for the

assistance of application level. In order to realize this problem, we investigate and develop a new compiler which is suitable for $\lambda$ computing and supports parallel application.

## 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 my studies. His excellent guidance and thoughtful advice that I could have through my studies are irreplaceable and invaluable my fortune and will help me in my life.

I am most grateful to Associate Professor Ken-ichi Baba at Osaka University for his much appreciated comments and support. All of my works would not have been possible without his worthwhile suggestions and encouragement.

I wish to express my sincere appreciation to Professors Koso Murakami, Makoto Imase, Teruo Higashino, Hirotaka Nakano, and Tetsuji Satoh of Osaka University, for their appropriate guidance.

I appreciate Associate Professor Noriyuki Fujimoto at Osaka University, Dr. Akira Okada at NTT Photonics Laboratory and Dr. Hiroaki Harai at National Institute of Information and Communications Technology for their useful advices and discussion.

I am also indebted to Associate Professor Naoki Wakamiya, Associate Professor Go Hasegawa, Assistant Professor Shin'ichi Arakawa and Assistant Professor Masahiro Sasabe at Osaka University for their helpful comments.

I want to heartily thank my friends and colleagues in the Department of Information Networking of Osaka University.

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

## References


[1] Lou Berger Ed., "Generalized Multi-Protocol Label Switching (GMPLS) Signaling Functional Description," *IETF RFC3471*, Jan. 2003.

[2] Hirohisa Nakamoto, Ken-ichi Baba, and Masayuki Murata, "Proposal and Evaluation of Realization Approach for a Shared Memory System in $\lambda$ Computing Environment," in *Proceedings of the forth International Conference on Optical Internet (COIN2005)*, pp. 90–95, May 2005.

[3] Eiji Taniguchi, Ken-ichi Baba, and Masayuki Murata, "Implementation and Evaluation of Shared Memory System for Establishing $\lambda$ Computing Environment," in *Proceedings of 10th OptoElectronics and Communications Conference (OECC2005)*, pp. 20–21, July 2005.

[4] Message Passing Interface available at `http://www-unix.mcs.anl.gov/mpi/`.

[5] Mai Imoto, Eiji Taniguchi, Ken-ich Baba, and Masayuki Murata, "Implementation and Evaluation of MPI Library with Globus Toolkit for Establishing $\lambda$ Computing Environment," in *Proceedings of 6th Asia-Pacific Symposium of Information and Telecommunication Technologies (APSITT 2005)*, pp. 421–426, Nov. 2005.

[6] Hideharu Amano, *Parallel Computers*. Shokodo, 1996. (in Japanese).

[7] Oudewijn R. Haverkort, *PERFORMANCE OF COMPUTER COMMUNICATION SYSTEMS*. WILEY, 1998.

[8] Masaaki Harada, *Probability Model*. McGraw–Hill, 1977. (in Japanese).

[9] Kazuki Joe and Jun Naito, "An Analytic Model for the Performance of the ASURA Cluster using a Semi–Markov Processing," *Technical Report of IPSJ* (ARC–1992–097), pp. 65–72, 1992. (in Japanese).

[10] John L. Hennessy and David A. Patterson, *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 1996.

[11] Gary S. Delp and David J. Farber and Ronald G. Minnich and Jonathan M. Smith and Ming-Chit Tam, "Memory As A Network Abstraction," *IEEE Network Magazine*, 1991.

[12] Hirohisa Nakamoto, “Proposal and Evaluation of Realization Approach for a Shared Memory System in $\lambda$ Computing Environment,” Master’s thesis, Osaka University, 2005.

[13] Gary S. Delp, Adarshpal S. Sethi, and David J.Farber, “An Analysis of Memnet: An Experiment in High-Speed Shared-Memory Local Networking,” in *ACM SIGCOMM Computer Communication Review*, vol. 18, pp. 165–174, ACM Press, 1988.

[14] Hirohisa Nakamoto, Ken-ichi Baba, and Masayuki Murata, “Shared memory access method for a lambda computing environment,” in *Proceedings of Optical Netwrok and Technology Conference 2004 (OpNeTech2004)*, pp. 210–217, Oct. 2004.

[15] William A. Wulf and Sally A. McKee, “Hitting the Memory Wall: Implications of the Obvious,” *Computer Architecture News*, vol. 23, no. 1, pp. 20–24, 1995.

[16] Norihisa Suzuki, Shigenori Simizu, and Nagatsugu Yamanouchi, *An Implementation of a Shared Memory Multiprocessor*. CORONA PUBLISHING, 1993. (in Japanese).

[17] Kiyofumi Tanaka, Takashi Matsumoto, and Kei Hiraki, “Quantitative Eavaluation of Scalable Directory Schemes in Hardware Distribured Shared Memory,” *Technical Report of IPSJ* (ARC–2000–129), pp. 7–12, 2000. (in Japanese).

[18] Akira Okada, Hiromasa Tanobe, and Morito Matsuoka, “Dynamically reconfigurable real–time information–sharing network system based on a cyclic–frequencyAWG and tunable–wavelength lasers,” in *Proceedings of 29th European Conference on Optical Communication (ECOC2003)*, 2003.

[19] Hiromi Okada, *Information Network*. Baifukan, 1994. (in Japanese).

[20] Akihiro Hashimoto, *Computer Architecture*. Shokodo, 1995. (in Japanese).

[21] Hideo Miyahara and Yuji Oie, *Computer Network*. Kyoritsu Shuppan, 1999. (in Japanese).

[22] Thomas DeFanti, Maxine Brown, Jason Leigh, Oliver Yu, Eric He, Joel Mambretti, David Lillethun, and Jeremy Weinberger, “Optical Switching Middleware for the OptIPter,” *IEICE Transaction on Communication*, vol. E86-B, pp. 2263–2272, Aug. 2003.