A New Network Interface with Distributed Memory

Hiroyuki Okuno[†], Yasushi Inoguchi[‡], Susumu Horiguchi[†]

2/Aug/2001
IS-RR-2001-018

†School of Information Science ‡Center for Information Science
Japan Advanced Institute of Science and Technology
Asahidai 1-1, Tatsunokuchi
Ishikawa, 923-1292, JAPAN

hirono@jaist.ac.jp, inoguchi@jaist.ac.jp, hori@jaist.ac.jp

ISSN 0918-7553

Abstract

In this paper, a new network interface with distributed memory is proposed for clustering WorkStation (WS) or Personal Computer (PC) to reduce the data transfer within a node of a cluster. Clustering WS or PC is expected to realize high performance computing (HPC). However, the limited bandwidth and high network latency restrict the transfer speed for huge amount of data among nodes in a cluster. Therefore, it is important for cluster system to reduce data transfer as much as possible. Data transfer speed among nodes of a cluster has been improved significantly in recent years using giga-bit network layer. On the other hand, the data transfer speed within each node of a cluster is still a bottleneck, thus it is important to reduce the data transfer within a node. In order to reduce the data transfer, out network interface has a memory on the interface. For effective data storing into the memory on network interface, two functions Put and Back are also defined and they reduce the data transfer between main memory and network interface. The simulation results of matrix multiplication indicate that data transfer of message communication can be reduced dramatically by using our new network interface.

1 Introduction

Clustering WS or PC is an effective approach to realize high performance computing[1][2]. Compare to massively parallel computers which are very expensive, clusters can be built using thousands of very cheap PC to achieve comparable computing performance. However, clusters have a problem of data transfer. The data transfer problem is twofold: inter-node and intra-node. Inter-node data transfer speed is improved by giga-bit networks such as Myrinet and Gigabit Ethernet[3]. On the other hand, intra-node data transfer still has no effective improvement.

There are several remarkable works about this problem. In [4], Minnich proposed an ATM network interface integrating into SIMM memory slot. By combine the network interface with higher-bandwidth memory slots, the interface achieved around 1Gbit throughput. Since SIMM is declining now, so it is not likely that the ATM network interface will become popular in the future. In [5], Tanabe proposed a network interface similar to MINI[4], but he used DIMM memory instead of SIMM memory in his new structure. Although he could reduce bus-bottleneck on data transfer between main memory and network interface, it requires particular device that restricted by memory slot specification. SCIMA[6] proposed on-chip memory that reduces data transfer between CPU and memory by including memory within CPU. Although the on-chip memory can reduce data transfer between CPU and memory, they did not discuss the problem about how to reduce data transfer between main memory and network interface. There are many results about how to improve performance of cluster, however, only little results available on how to reduce data transfer between main memory and network interface. Although several network interface have been proposed in past studies for this problem[7][8], these network interface only have little buffer. Thus they still need to transfer data between main memory and network interface for every message communication.

In this paper, we propose a new network interface for distributed memory cluster system. The network interface has a memory and CPU in each node an access the memory as same as the memory on the CPU's board. To use memory on network interface as extension of main memory and store data for message communication, it can reduce data

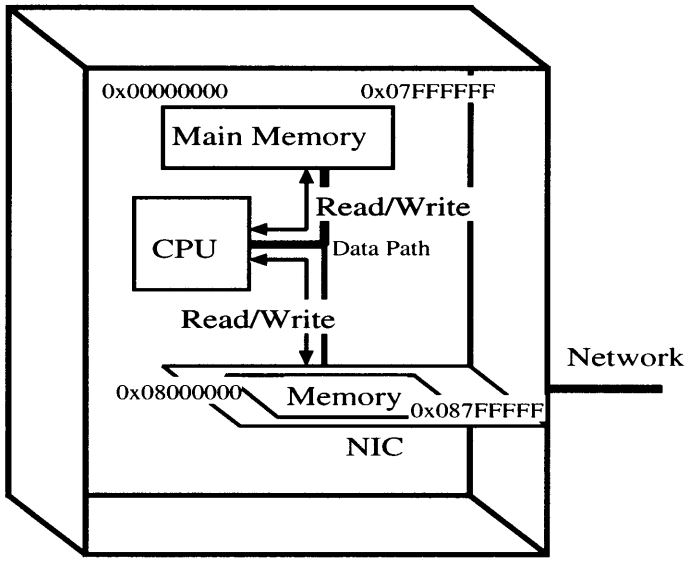

Figure 1: System overview of network interface using distributed memory

transfer between main memory and network interface. To store data into memory of network interface effectively, we propose two data transfering functions Put and Back. Sending data of message communication using Put, it can reduce data transfer from main memory to memory of network interface on message communication. The Back function is used for returning data to main memory, it can reduce data transfer from memory of network interface to main memory on CPU calculation.

The rest of this paper is organized as follows. Section 2 describes a new network interface, data transfer protocol between main memory and memory of network interface and, its trade-off. In Section 3, the performance of the proposed network interface is verified by simulation of matrix multiplication. Finally, we summarize the paper and conclude our remarks in Section 4.

2 Network Interface Using Distributed Memory

2.1 Outline of Network Interface

In this section, we propose a new network interface to reduce data transfer between main memory and memory of network interface. Fig.1 shows the proposed network interface, where memory is implemented and is used like main memory. As shown in Fig.1, memory

implemented on network interface is assigned memory address space on the heels of main memory address. Therefore, memory implemented on network interface can be used as a part of main memory. Hereafter, we use the term "NIC memory" to refer the memory implemented on network interface, and "NIC" to the proposed network interface. And the term "Old-NIC"is used to refer the usual network interface.

We assume that NIC is equipped on PCI-bus because PCI-bus can provide more widely usage compared to that of MINI[4] or MEMOnet[5]. When accumulation technology is considered, it is reasonable to assume that the memory size of NIC under consideration varies from 2 MB to 64 MB. In this paper, we assume that NIC memory size is 8MB, and data can be transfered bi-directionally between main memory and NIC memory.

The merits of our approach are presented as follows. Since we use NIC memory as a part of main memory, same data will not exist in main memory and NIC memory simultaneously. This feature means that we need nor to consider the data coherency problem between main memory and NIC memory. Moreover, unless process invoke data transfer between NIC memory and main memory, the data could be exist in NIC memory. That is, there is no need to transfer data from main memory to NIC memory at message communication, while the Old-NIC, which uses buffers need to copy the data from main memory to network interface buffer in every message communication. Therefore our proposed NIC could be expected to reduce data transfer of message communication. However, to reduce data transfer of message communication using proposed NIC, it needs data transfer functions. In the next part, we describe proposed data transfer protocol.

2.2 Data Transfer Protocol Between Main Memory and NIC Memory

To realize our proposed NIC and reduce data transfer of message communication, we propose two functions Put and Back that enable us to transfer data between main memory and NIC memory. Fig. 2 illustrate the functions of Put and Back which are described as follows.

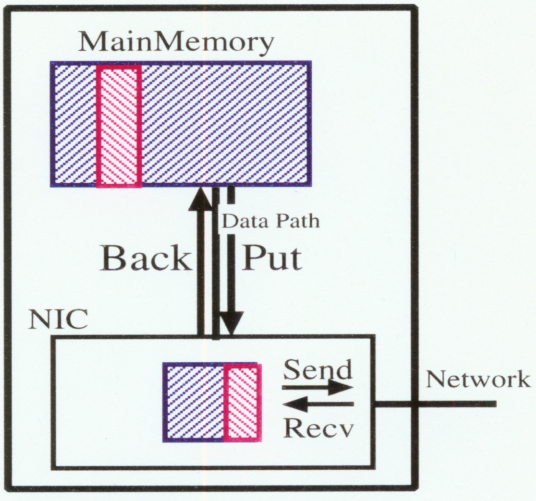

Figure 2: Concept of Put and Back function

2.2.1 Put: Transfering data from main memory to NIC memory

The function Put transfers data from main memory to NIC memory. It is automatically invoked in message communication if data is not exist on NIC memory. We illustrate the operation of Put function in Fig.3. In message communication, if data exists on main memory (B1), then message communication process is interrupted and data are read from main memory (B2). After reading memory, the data are transfered to NIC (B3) and wrote into NIC memory (B4). Finally, message communication is resumed and a message is sent to destination nodes from NIC memory or received from source nodes to NIC memory (B5). On the other hand, if data already exists in NIC memory (A1), a message is immediately sent to destination nodes from NIC memory or received from source nodes to NIC memory (A2).

2.2.2 Back: Transfering data from NIC memory to main memory

In contrast to Put function as mentioned above, The Back function transfers data from NIC memory to main memory. The Back function is also invoked automatically at CPU calculation if data do not exist in main memory. The operation of Back is illustrated in Fig.4. When CPU is calculating and if data exist in NIC memory (D1), the calculation process is then interrupted and data are read from NIC memory (D2). After NIC memory

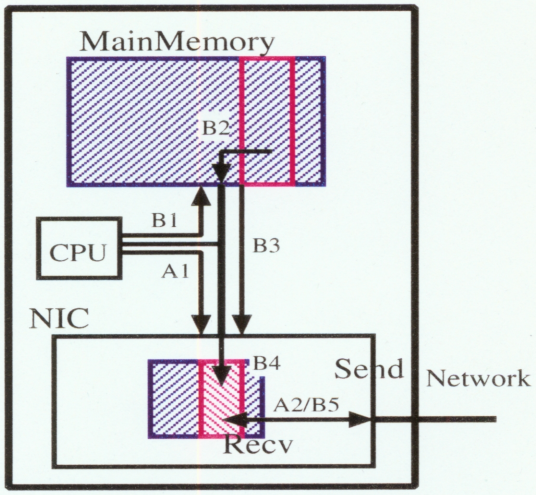

Figure 3: Data flow on Put operation

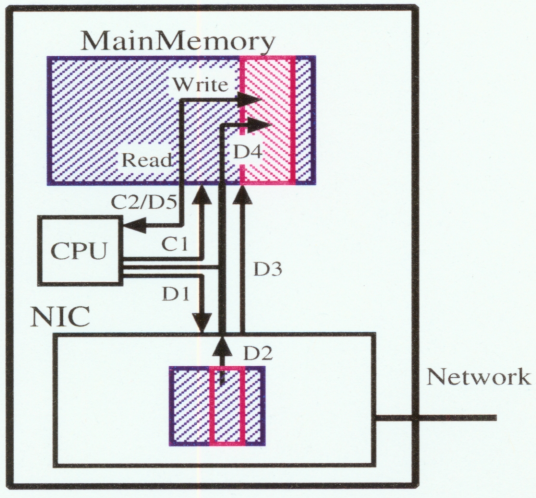

Figure 4: Data flow on Back operation

read data from NIC memory, it will transmit data to main memory (D3) and write data into main memory (D4). At last, calculation is resumed and accessed to main memory (D5). If data already exist in main memory (C1), data is normally read from memory or write to main memory (C2). Returning data that messaged only one time from NIC memory to main memory using this Back function, CPU can access to main memory effectively without accessing from CPU to NIC memory.

Both Put and Back involve a problem that they cannot cope with the lack of memory. To perform Put or Back, enough memory space should be ensured to use page replacement algorithm such as LRU. But our purpose here is to explore data transfering performance between main memory and NIC using Put and Back. Thus it needs further consideration including circumstantial specifications.

To migrate data for message communication into NIC memory using Put, we can send message to destination nodes from NIC memory directly or receive message from source nodes to NIC memory directly without transfering data to main memory. However, this method separates main memory into two and placed in two different places. Therefore we need to discuss the trade-off listed bellow.

- Access speed between CPU and main memory is fast.

  $\leftrightarrow$ Access speed between CPU and NIC memory is slow.

- Message communication from NIC memory is fast.

  $\leftrightarrow$ Message communication from main memory is slow .

In the next section, we examine the proposed NIC in simulation environment and discuss these two trade-off.

3 Evaluation of NIC Memory in Simulation

3.1 Matrix Multiplication and Conditions of Simulation

To evaluate proposed NIC, we did matrix multiplication in simulation environment. At first, we describe the multiplication technique of matrix ($C = A \times B$). A node contains subblocks of matrix C,A, and B. Matrix is partitioned as shown in Fig.5. In Fig.5, *size*

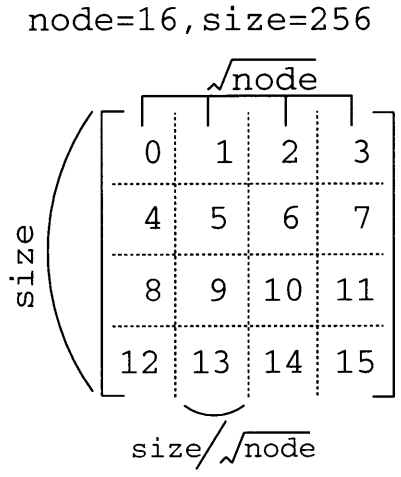

Figure 5: Matrix subblock partitioning on 256x256 matrix by 16 nodes

is 256 and *node* is 16. Therefore, size of subblock is calculated as follows: $256/\sqrt{16} \times 256/\sqrt{16} = 64 \times 64$. Each node has only partitioned subblock of C, A and B, as mentioned above, calculation needs to exchange subblock A and B using message communication. First each diagonal node (in Fig.5, it is 0, 5, 10, 15) multicasts its own subblock A to other node in same row (if node 0, it multicasts to node 1, 2, 3). After multicasting A, each node multiplies A and B. Then each node send B to the upper column node, except top column node send to the bottom column node. Calculation repeats above message communication in $\sqrt{node}$ iterations. We did this matrix multiplication in the following conditions: Matrix size is 256 × 256, the number of executed node is either 4, 16 and 64.

In our simulation, we recorded memory accesses of data per node. Memory accesses are recorded in both read and write accesses to main memory or NIC memory. We recorded the number of accesses in following two cases. One uses only Put function, and the other uses both Put and Back. Since we recorded memory accesses only in matrix calculation interval, we didn't take the record of another interval such as initialization of matrix multiplication. Simulation using only Put function was executed to compare it with using both Put and Back. In simulation, we assume that main memory has enough size to store whole data. Therefore we don't consider about page-out and page-in of data.

3.2 Definition of Memory Access State

We define memory accesses state to main memory or NIC memory. The memory access state are classified into four as follows.

- Accesses to main memory
  - When CPU reads from main memory, data exist on there or not.

    The former is defined as RMH (Read Memory Hit), the latter is defined as RMF (Read Memory Failed).
  - When CPU writes to main memory, data exists on there or not.

    The former is defined as WMH (Write Memory Hit), the latter is defined as WMF (Write Memory Failed).

  In these accesses, if data exist in NIC memory, then access to main memory is interrupted and the Back function is invoked. Then data is transfered from NIC memory to main memory. When the Back function is performed, read from main memory or write to main memory is resumed. We regard the Back operation of transfering data to main memory as WMF and it is performed after accessing the main memory as RMH or WMH.

- Accesses to NIC memory
  - When send a message, it exist in NIC memory or not.

    The former is defined as RNH (Read NIC Hit), the latter is defined as RNF (Read NIC Failed).
  - When receive a message, it exist in NIC memory or not.

    The former is defined as WNH (Write NIC Hit), the latter is defined as WNF (Write NIC Failed).

  In these accesses, if data exist in main memory, then message communication process on NIC is interrupted and Put function is invoked. After Put transfer data from main memory to NIC memory, message communication process is resumed. Similar

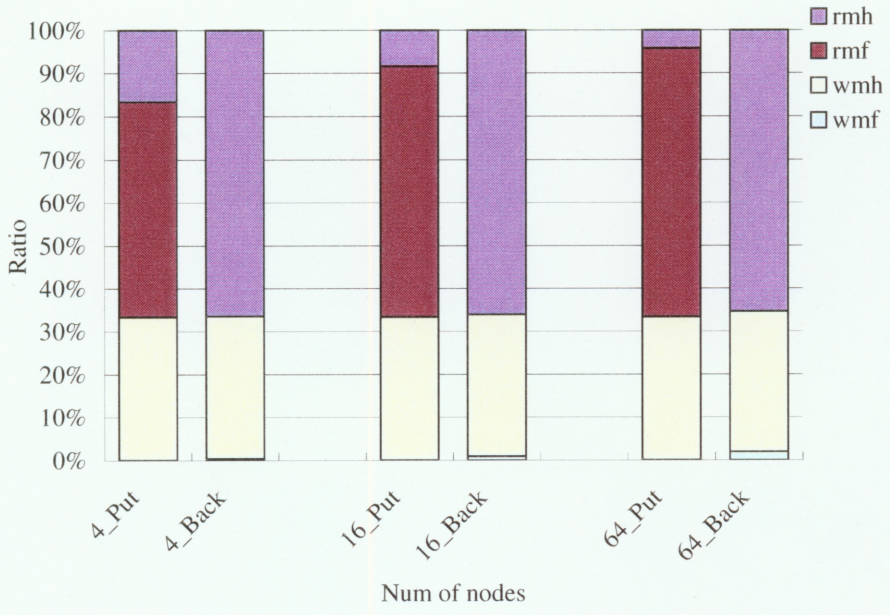

Figure 6: Access hit/miss ratio to main memory

to the Back function, we regard Put operation of transfering data to NIC memory as WNF and it is performed after accessing the NIC memory as RNH or WNH.

To discuss trade-off as described in the previous section, we need to make how memory accessed clear. From the next section, we describe about it.

3.3 Result of Main Memory Accesses

The result of main memory access is shown in Fig.6. In Fig. 6, each bar chart is separated in pair by executed number of nodes. In a pair of bar chart, left bar chart shows result of using only Put and right bar chart shows result of using both Put and Back. Each item in a bar chart indicates memory access state as we mentioned in previous section 3.2. From Fig. 6, we found that when use only Put, RMF accounts for more than 50 percent of memory access. In matrix multiplication, each node have only subblocked matrix and needs to exchange using message communication. Consequently, matrix A and B have to read from NIC memory on calculation and result in such highly RMF ratio. On the other hand, when both Put and Back are used, RMH accounts for 70 percent of memory access. The reason of this result is that the Back transfers matrix A and B to main memory

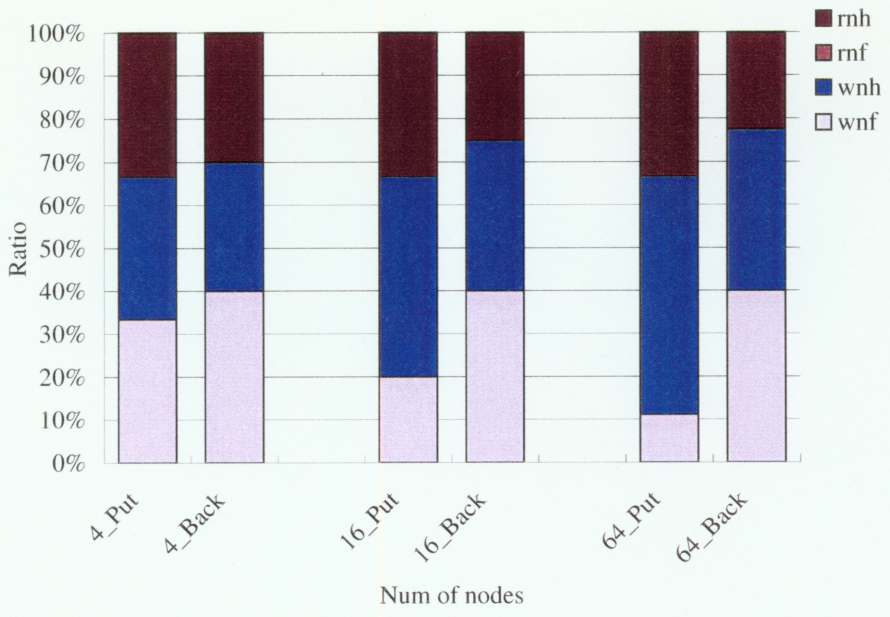

Figure 7: Access hit/miss ratio to NIC memory

before multiplication and lead memory access to RMH.

These results indicate that when Put and Back are used together, slow access between CPU and NIC memory is permuted fast access between CPU and main memory. Thus, it can reduce data transfer between main memory and NIC memory.

3.4 Result of NIC Memory Accesses

Record of NIC memory accesses is shown in Fig. 7. Each bar chart in Fig. 7 is separated in pair by executed number of nodes, and each left bar of pair indicates result of using only Put and each right bar of pair indicates result of using both Put and Back. Each items in a bar chart indicate access states of NIC memory. Since whole data to be communicated exist in NIC memory, results of using only Put indicate that the WNH ratio increased in proportion to nodes. Because number of message communication per node increase in proportion to increase number of nodes. On the other hand, results using both Put and Back function indicates that WNF ratio is increased compared with results of using only Put. Since matrix multiplication algorithm mutually execute CPU calculation and message communication, Put and Back are invoked and transfer data whenever these

Table 1: Increased number of Put, Back, RMH in matrix multiplication using Put and Back and elements of matrix subblock

Num of node	4	16	64
Num of Put	+1	+5	+13
Num of Back	+3	+7	+15
Num of RMH	+6,307,840	+1,839,104	+492,544
Elements of a matrix subblock	16,384	4,096	1,024

process executed. This means that data transfer between main memory and NIC memory performed many times. Although it is confirmed that using Put can reduce data transfer between main memory and NIC memory, from above results, Back increases WNF ratio which induce slow memory access in proportion to nodes. Next we shall discuss about effectiveness of Back in detail.

3.5 Effectiveness of Back

We proposed the Back function in subsection 2.2. By returning the data to main memory using the Back function, we can access the data from main memory again. As we have mentioned earlier, however, there is possibility that repeated data transfer by Put and Back could lead to bus-bottleneck and results decline in performance. To examine the amount of data transfered by Put and Back in matrix multiplication and accessed as RMH in matrix multiplication, respectively, we can see whether utilization of the Back function leads to bus-bottleneck. In this paragraph, we discuss the effectiveness of the Back function from the point of view of the amount of data in matrix multiplication.

First we show an increase in the number of Put, Back and RMH in matrix multiplication as a result of using only Put in Table 1. As shown in this table, as the number of nodes is increased, the number of Put and Back are also increased. This causes additional data transfer between main memory and NIC memory. Since the number of RMH are counted by elements of matrix subblocks, we can confirm a smaller increase n the number of RMH as the number of nodes increases in Table 1.

Assuming the matrix data type is floating point, from Table 1, we can get the amount of data in matrix multiplication in Table 2. According to Table 2, total data size by Put and

Table 2: Data size by Put and Back and data size of RMH (unit: KByte)

Num of node	4	16	64
Data size increased by Put	64	80	52
Data size increased by Back	192	112	60
Total data size by Put and Back	256	192	112
Total data size of RMH	24,640	7,184	1,924

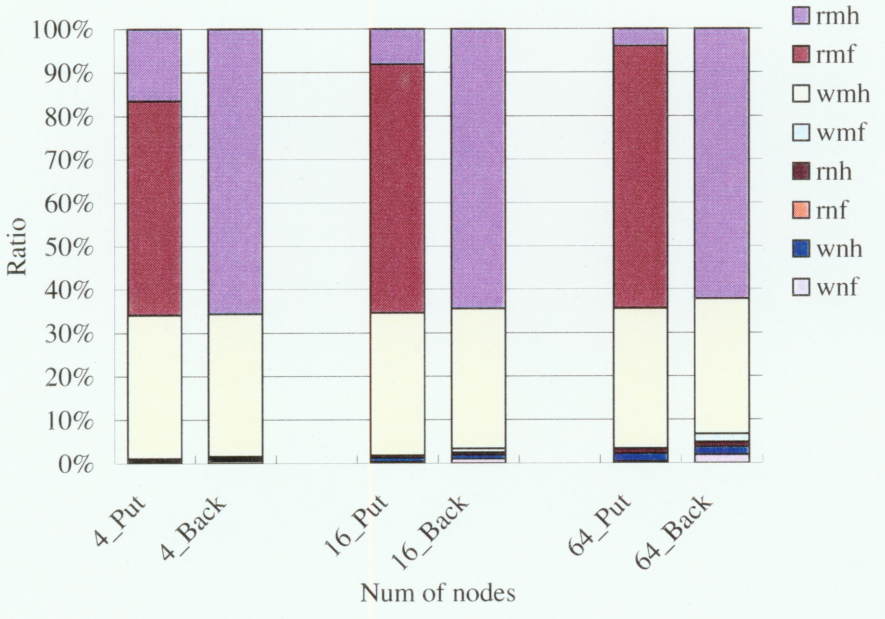

Figure 8: Total access hit/miss ratio to main/NIC memory on 256×256 matrix multiplication

Back is only 256KByte on 4 nodes. In contrast to total data size by Put and Back, total data size of RMH is extremely large. Since the Back returns the data to main memory and following data accesses to main memory result in RMH, increased numbers of RMH can be obtained using the Back. Therefore, total data size of RMH can be regarded as total saved data transfer size between main memory and NIC memory. From Table. 2, using both Put and Back can save large amount of data transfer between main memory and NIC memory compared to that of Pub and Back.

We show Fig.8 as a summary of Fig. 6 and Fig. 7. As shown in Fig.8, newly recorded ratio of WMF and WNF are very small compared with the increased ratio of WMH obtained using the Back. From Fig.8, we also confirmed that it seems reasonable to

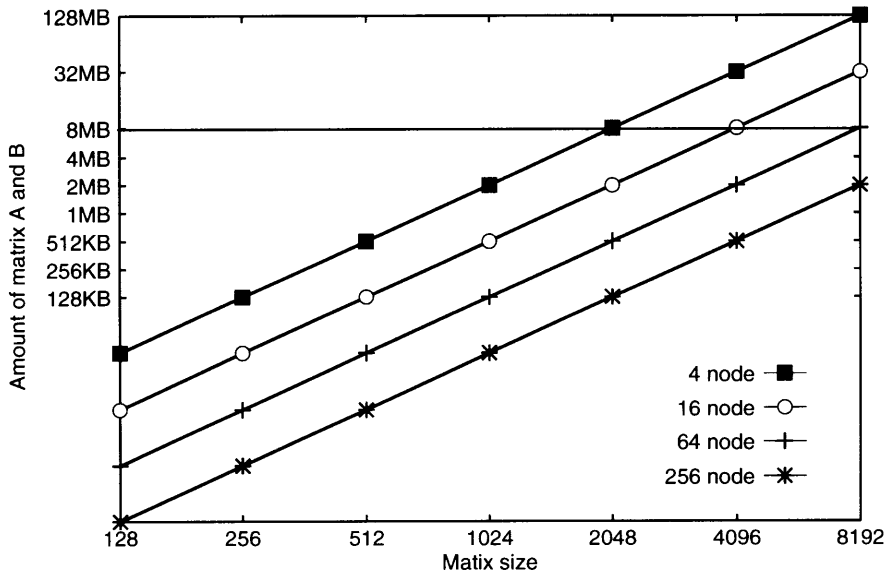

Figure 9: Memory size of subblock matrix A and B concerned with node and matrix size variation

suppose that the decline in performance by the Back is negligible. From the results mentioned above, we confirmed that the Back can save extra data transfer between main memory and NIC memory without performance corruption.

3.6 Discussion About NIC Memory Size

In the above simulation, we fixed NIC memory size as 8MB. This paragraph discusses suitability of NIC memory size. The data which transferred to NIC memory on matrix multiplication are subblock of matrix A and B. Consider data type of matrix as floating point, total memory size of subblock A and B on each node is calculated as follows: "(the elements of matrix subblock) $\times$ (size of data type) $\times 2$" (there 2 means two matrix: A and B). For example, the number of node is 16 and matrix size is 256, then memory size is: $(256/\sqrt{16})^2 \times 4 \times 2 = 32,768$ byte. We show these memory sizes variation concerned with node and matrix size in Fig.9. According to Fig.9, it shows that even 4 node cluster can execute size of $2,048 \times 2,048$ matrix multiplication within 8MB NIC memory. Moreover, it indicates that 256 nodes could be executed up to $16,384 \times 16,384$. From this discussion, it is clear that 8MB of memory is enough to perform large scale matrix multiplication.

4 Conclusion

In this paper, we proposed a new NIC using distributed memory containing a memory referred from CPU and evaluated its performance by simulation. CPU in each node access the memory on NIC as an extension of main memory and stores the data for message communication. From the performance evaluation of NIC, it is confirmed that proposed NIC could reduce data transfer between main memory and NIC. There was a trade-off between an effect of the Back function and extra data transfer with the Back function. As the result of simulation, an effect of the Back function was superior to extra data transfer with the Back function and NIC could reduce data transfer time.

References

[1] http://now.cs.berkeley.edu/

[2] http://www.llnl.gov/asci/

[3] Scott Pakin, Mario Lauria and Andrew Chien "High Performance Messaging on Workstations: Illinois Fast Messages (FM) for Myrinet", Proceedings of Supercomputing '95, San Diego, California (1995).

[4] Ron Minnich, Dan Burns and Frank Hady "The Memory Integrated Network Interface", IEEE Micro, Vol.15, No.1, (1995.2)

[5] Noboru Tanabe, Junji Yamamoto and Tomohiro Kudoh "MEMnet : Network interface attached on memory slot", In IPSJ SIG Notes 99-ARC-134 (SWoPP'99), pp.73-78 (1999.8) (In Japanese)

[6] Hiroshi Nakamura, Masaaki Kondo, Hideki Okawara and Taisuke Boku "SCIMA: A new architecture for High Performance Computing" IPSJ Transactions on High Performance Computing Systems, Vol.41, No.SIG5 (HPS 1), pp.15-27 (2000) (In Japanese)

[7] James V.Lawton, John J. Brosnan, Morgan P. Doyle, Seosamh D. ORiordain, Timothy G. Reddin "Building a High-performance Message-passing System for MEMORY CHANNEL Clusters", Digital Technical journal Vol.8, No.2 (1996)

[8] M. Fillo, R. B. Gillett "Architecture and Implementation of MEMORY CHANNEL 2", Digital Technical Journal, Vol.9, No.1 (1997)