KMITL Sci. J. Vol. 5 No. 1 Feb. 2005

SPACE-EFFICIENT PARALLEL BITONIC SORTING ON A CLUSTER OF PCS

Jureeporn Boonniyom[*], Phairoj Samutrak, and Jeeraporn Srisawat

Department of Mathematics and Computer Science
Faculty of Science
King Mongkut's Institute of Technology Ladkrabang(KMITL)
Ladkrabang, Bangkok 10520, THAILAND

ABSTRACT

This paper presents the space-efficient parallel Bitonic sorting for a very large data set on a cluster of PCs by using MPI. Recently, most studies have focused on the theoretical approach of the parallel Bitonic sorting on shared-memory or distributed-memory parallel computers. As a combination of theoretical and practical approach, we are interested to study and implement the parallel Bitonic-sorting on a cluster of PCs with efficient-space and efficient-communication overhead. In such cluster environment, the system performance of our parallel Bitonic sorting (NBS) and existing Bitonic sorting (BS) have been compared in terms of response time, speedup, and efficiency. In experimental results, our space-efficient parallel Bitonic sorting yielded similar results to those of the parallel Bitonic MPI-based sorting, while the space of our method was improved up to 50%.

KEYWORDS: Parallel Bitonic sorting, efficient space, efficient communication, MPI (Message Passing Interface), a cluster of PCs.

[*] Tel: 0-6803-9858, E-mail s7063605@kmitl.ac.th, onenoi@hotmail.com

1. INTRODUCTION

Sorting is one of the most common operations performed by computers in many applications. However, sequential sorting on very large data sets is time consuming. Therefore parallel Bitonic sorting [1], one of the most popular parallel sorting method, is introduced because of its fast processing. It's time complexity is only $O(\text{Log}^2 N)$, where N is a number of data being sorted on P processors ($P = N$). In the past ten years, many studies were proposed in order to improve communication overhead of parallel Bitonic sorting [5], [6], [7] and also adapted them to specific parallel machines. There are two main theoretical approaches that proposed the efficient-communication parallel Bitonic sorting: 1) one is based on shared-memory parallel systems [5], [7] and 2) another one is based on distributed-memory parallel systems [6]. Recently, more practical sorting methods on cluster(s) of computers, a class of distributed-memory parallel systems, were introduced as coarse-grained computing ($P < N$), which are the merge-sort-based parallel sorting on a cluster of heterogeneous PCs [2] and the quick-sort-based parallel sorting on clusters of SMPs (symmetric multiprocessors) [4].

In this paper, we present a combination of theoretical and practical parallel Bitonic-sorting with efficient space on a coarse-grained cluster of PCs ($P < N$) by using MPI (Massage Passing Interface) standard [3], [8], [9]. We also present the investigation of the system performance (i.e., response time, speedup, and efficiency) of our space-efficient parallel Bitonic sorting (NBS), compared to that of existing Bitonic sorting (BS) on a cluster of PCs.

The remainder of this paper is organized as follows. Section 2 gives the fundamental definition and original parallel Bitonic sorting algorithm. Section 3 proposes our space-efficient parallel Bitonic sorting method for a cluster of PCs and Section 4 shows the investigated results of the system performance evaluation. Finally, Section 5 presents the conclusion of this study and discusses our future study.

2. MATERIALS AND METHODS

2.1 Bitonic Sorting

<u>Definition 1</u>: A Bitonic sequence is a sequence of a data set $(a_0, a_1, a_2, \ldots, a_i, \ldots, a_{N-1})$, where $a_0, a_1, \ldots, a_i$ is a monotonic increasing sub-sequence and $a_{i+1}, a_{i+2}, \ldots, a_{N-1}$ is a monotonic decreasing sub-sequence ($0 \le i \le N-1$).

An instance of a Bitonic sequence (N=16) is $\underline{4\ 5\ 6\ 8\ 10\ 15\ 20\ 30}\quad\underline{28\ 25\ 20\ 15\ 12\ 9\ 3\ 1}$.

To perform Bitonic sorting on a Bitonic sequence, a single compare-exchange step can split a single bitonic sequence (of n values) into two bitonic sequences (of n/2 values), $<\min(a_i, a_{n/2+i})>$ and $<\max(a_i, a_{n/2+i})>$; $i = 0, 1, \ldots, n/2 - 1$. However, for any sequence, transformation step (Fig. 1) is required in order to create a Bitonic sequence.

<u>Definition 2</u>: A Bitonic Sort Network is a sorting of N elements (of any sequence) on P processors (where P=N) using $\log_2 N$ stages in $O(\log^2 N)$, as depicted in Fig.1.

where $\uparrow$ represents decreasing sub-sequence ($<\min(a_i,a_{n/2+i})>$) and
$\downarrow$ represents increasing sub-sequence ($<\max(a_i,a_{n/2+i})>$)

An example of Bitonic sort network is shown in Fig.1.

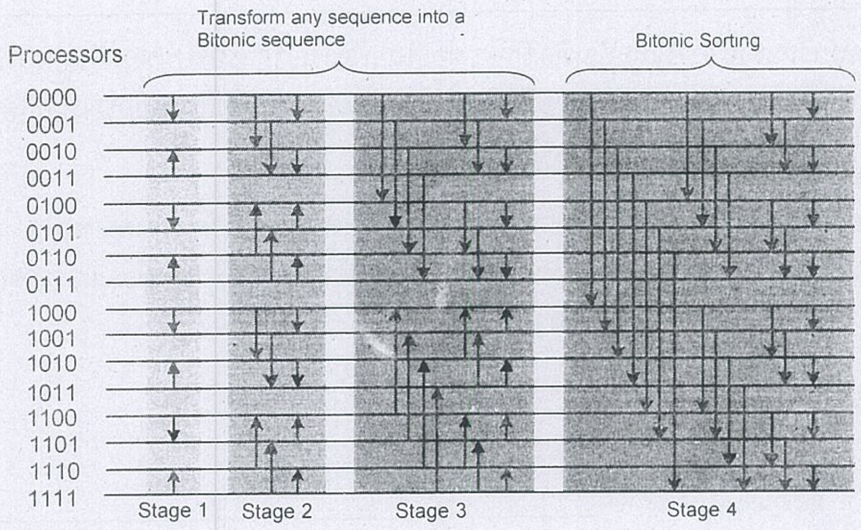

Fig. 1. Bitonic sort stages with 16 processors.

A Bitonic sorting algorithm for any sequence is presented in (C-like program) Algorithm 1.

```
Global      d: Distance between elements being compare
Local       a: One of the elements to be sorted
               t: Element retrieved from adjacent processor
Begin
      for l = 0 then m-0 do   // $m = \log_2 N$
            for J = l downto 0 do
               $d = 2^J$
               // For All Processor
               for   all Processor k where 0 <= k <= 2m-1 pardo
                        if k mod 2d < d then
                           t =[k + d]a
                           if  k mod  2l+2 < 2l+1  then
                        [k+d]a =  max(t, a) //sort low to high
                                 a =  min(t, a)
                           else
                                 [k+d]a =  min(t, a) //sort high to low
                                 a =  max(t, a)
                           end if
                        end if
               end for all
            end for J
      end for l
end
```

Algorithm 1: Bitonic Sorting Algorithm (P = N).

The inter-process communication in the above Bitonic sorting algorithm is called compare-exchange operation, depicted in Fig.2. Initially, each processor has one element in its local memory (Fig2(a)). Next, each communication pairs of processors (P_i and P_j) send a copy of its element to and receive a copy of another element from its partner (Fig.2(b). Finally (Fig.2(c)), P_i stores min (a_i, a_j) and P_j stores max (a_i, a_j).

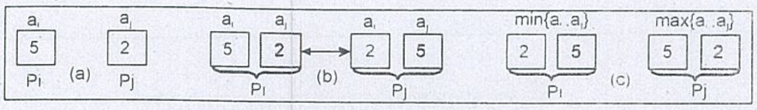

Fig. 2. Compare-Exchange Operation.

In case of medium-grained or coarse-grained ($P < N$), first each processor is responsible for sorting N/P elements locally. Then, these P processors need to communicate in order to perform Bitonic sorting for the larger sorted sequences. The inter-process communication now is called compare-split operation, illustrated in Fig.3. In this example, there are 2 processors (P=2) and 10 elements (N=10). Initially, each P_i is responsible to sort 5 elements (N/P) locally (Fig.3(a)). Then, P_i and P_j need to communicate to form the Bitonic sequence (Fig.3(b)). Processor P_i and P_j send a copy of its sub-sequence (of 5 elements) to and receive a copy of another sub-sequence (of 5 elements) from its partner (Fig.3(c). Finally (Fig.3(d)), after performing locally merge sort, P_i stores the minimum sub-sequence and P_j stores the maximum sub-sequence.

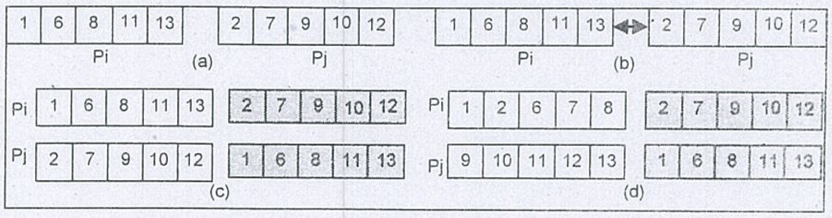

Fig. 3. Compare-Split Operation.

2.2 Space-Efficient Parallel Bitonic Sorting

We propose in this paper to reduce the redundant copying data of the existing parallel Bitonic sorting in an efficient way. In practical, we apply our efficient-space parallel Bitonic sorting on a cluster of PCs using C programming language with MPI standard. In the implementation of our parallel Bitonic sorting using MPI, the program consists of P workers (or computing nodes). Each processor (P_i) in the worker group receives N/P elements of the unsorted sequence, sorts them locally, and then communicates to its partner to merge the result.

The MPI code (for N elements on P processors) and an example of work load (N/P) for each p_i (when N = 10 and P = 2) is illustrated as follows:

```
Start
 MPI_Init(&argc, &argv);
 MPI_Comm_size(MPI_COMM_WORLD, &p);
 MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

Generate Data
 MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
 srand(my_rank);
 for (i = 0; i < list_size; i++)
     local_list[i] = rand() % KEY_MAX;

Local Sort
 qsort(local_keys, list_size, sizeof(KEY_T), int(*)(const void*, const void*))(Key_compare));

Bitonic Sort (BS)
 if ((my_rank & and_bit) == 0) // increase
              Par_bitonic_sort_incr(list_size, local_list, proc_set_size, MPI_Comm);
 else // decrease
              Par_bitonic_sort_decr(list_size, local_list, proc_set_size, MPI_Comm);

Partner Communication (send and receive data)
 MPI_Comm_rank(comm, &my_rank);
    proc_set_dim = log_base2(proc_set_size);
    eor_bit = 1 << (proc_set_dim - 1);
    for (stage = 0; stage < proc_set_dim; stage++) {
       partner = my_rank ^ eor_bit;
        if (my_rank > partner) {
          // OR  my_rank < partner for Par_bitonic_sort_decr
          MPI_Send(&local_list.list_size, key_mpi_t, partner, tagin, comm);
        }
        else {
          MPI_Recv(&temp_list.list_size, key_mpi_t, partner, tagin, comm, &status);
          Merge_split(list_size,temp_list, local_list, LOW, partner,comm);
      // Merge_split(list_size,temp_list, local_list, HIGH, partner, comm);
       }
       MPI_Send(temp_list[0],list_size, key_mpi_t, partner, tagout, comm);
       MPI_Recv(&local_list[0],list_size, key_mpi_t, partner, tagout, comm, &status);
 }

Low and High Value
 if (which_keys == HIGH)
     Merge_list_high(list_size, local_list, temp_list);
 else
     Merge_list_low(list_size, local_list, temp_list);
```

Algorithm 2: Space-efficient parallel Bitonic sorting algorithm (P < N).

In our coarse-grained approach (P < N), first each processor (P_i) is responsible for sorting N/P elements locally via quick sort in O($\log^2$ N/P). Similar to existing Bitonic sorting, corresponding pairs of processors need to communicate in order to perform parallel Bitonic sort to form the larger sorted sequences, except we introduce here the efficient-space communication, called the efficient compare-split operation, illustrated in Fig 4.

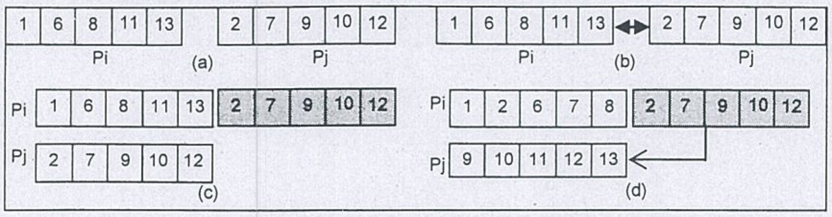

Fig. 4. Our Efficient Compare-Split Operation.

In order to compare to the existing approach, we use the same example as illustrated in Fig.3. Initially, each P_i (i = 1, 2, ..., P) is responsible to sort 5 elements (N/P = 5) locally (Fig.4(a)) and then P_i and P_j need to communicate to form the Bitonic sorting. In our approach, only processor P_j (where j > i) send a copy of its sub-sequence (of N/P elements) to its partner P_i and wait for the result (Fig.4(b)). In our case (Fig.4(c), there is no extra copy of N/P elements in processor P_j (as required in existing approach (Fig.3(c)), and hence we can save space up to 50%. However, while P_j is waiting, only P_i is responsible to perform locally merging the sorted sub-sequence and splitting it into two sub-sequences. Then, P_i stores the minimum sorted sub-sequence and send the maximum sorted sub-sequence back to P_j (Fig.4.d). Clearly, the computing and communication time of our approach are the same as those of the existing parallel Bitonic sorting method.

3. SYSTEM PERFORMANCE EVALUATION

In the system performance evaluation, we implement our space-efficient parallel Bitonic sorting (NBS) on a cluster of PCs. In our cluster environment, the system consists of 10 CPUs residing in 5 computer units (2 CPUs/unit). All computers in this cluster system are homogeneous, each of which is 2.4 GHz Xeon (1 GB RAM) and connected via a high speed 1000 Mbps LAN (Fig.5).

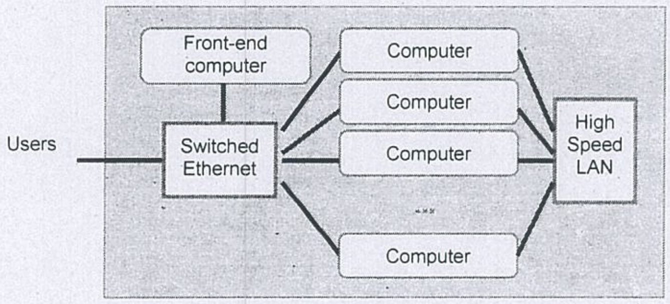

Fig. 5. A cluster of PCs environment.

A number of experiments were performed to investigate and compare our space-efficient parallel Bitonic sorting (NBS) and the existing parallel Bitonic sorting (BS) on such a cluster system. During experiments, many parameters (i.e., N (no. of elements), P (no. of processors)) were varied to investigate response time (T), speedup ($S = T_1/T_P$), and efficiency ($E = S/N$), where T_1 (s) is the execution time of running sequential program on one processor and T_p (s) is the execution time of running parallel program on P processors.

Table 1 showed the response time (T), speedup (S), and efficiency (E) of the BS sorting and our NBS sorting, where N = 10, 20, 40, 60, 80, 100 millions and P = 1, 2, 4, 8 processors. In all tested cases, the BS and NBS performed the comparable results, while our NBS can save space up to 50%. In particular, more than one half of these results, the NBS sorting yielded improved results than those of the BS sorting.

Table 1: Experimental Results of the BS and NBS sorting.

N	P	T1(s)	Tp (s) BS	S (BS)	E (BS)	Tp(s) NBS	S (NBS)	E (NBS)
10	2		4.79	1.29	0.64	4.89	1.26	0.63
Million	4	6.17	4.24	1.46	0.36	4.04	1.48	0.36
	8		4.07	1.52	0.16	3.87	1.59	0.20
20	2		8.98	1.38	0.69	8.97	1.38	0.69
Million	4	12.4	7.8	1.59	0.40	7.6	1.63	0.41
	8		8.77	1.41	0.18	8.47	1.46	0.18
40	2		18.64	1.36	0.68	17.64	1.43	0.72
Million	4	25.27	15.6	1.62	0.40	15.8	1.60	0.40
	8		15.44	1.64	0.20	14.84	1.70	0.21
60	2		28	1.38	0.69	28.2	1.37	0.69
Million	4	38.73	23.31	1.66	0.42	23.91	1.62	0.40
	8		22.97	1.69	0.21	22.87	1.69	0.21
80	2		37.38	1.11	0.59	38.08	1.16	0.58
Million	4	44.03	30.58	1.44	0.36	30.38	1.45	0.36
	8		67.7	0.65	0.08	67.5	0.65	0.08
100	2		47.05	1.19	0.60	46.65	1.20	0.60
Million	4	56.03	38.81	1.44	0.36	38.51	1.45	0.36
	8		105.23	0.53	0.07	104.87	0.53	0.07

For setting N = 50 million elements and varying P = 1, 2, 4, and 8, the investigated response time, speedup, and efficiency were illustrated in Fig. 6, 7, and 8, respectively. When increasing P to 2, the response time of both methods (Fig. 6) were improved by about 30% and their efficiency was approximately 0.72, closed to 1.0 (an ideal case). However, the response time was not improved when $P > 8$ for N = 50. Similar experimental results, as shown in response time, were presented for speedup (Fig. 7) and efficiency (Fig. 8) of both sorting methods for N = 50 million elements.

For using P = 4 processors and varying N = 1, 10, 20, 30, 40, 50, 60, 70, 80, 90, and 100 million elements, the investigated response time, speedup, and efficiency were illustrated in Fig. 9, 10, and 11, respectively.

When increasing the number of elements (N), our NBS sorting yielded the improved speedup and efficiency. Especially when N = 1, 10, 20, 30, 50, the speedup of the NBS sorting (Fig. 10) were improved up to 1.8 and also its efficiency (Fig. 11) were improved up to 0.45.

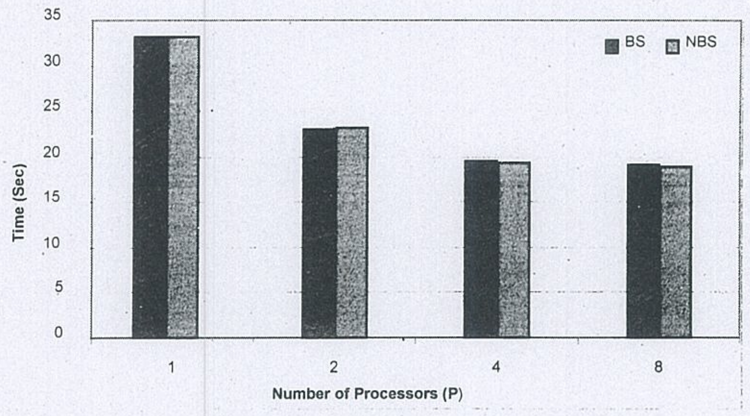

Fig. 6. Responses time of NBS and BS sorting (N = 50).

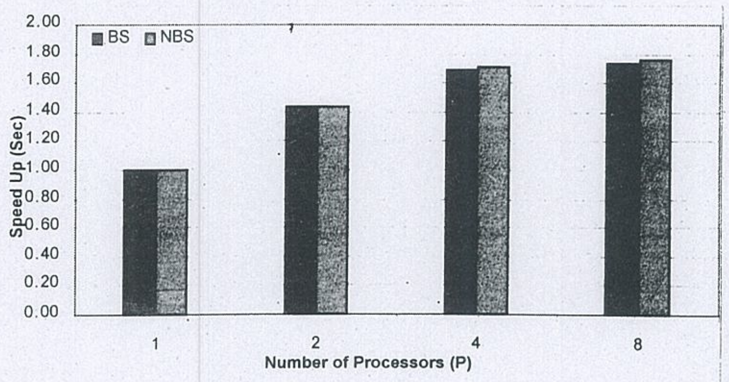

Fig. 7. Speedup of NBS and BS sorting (N = 50).

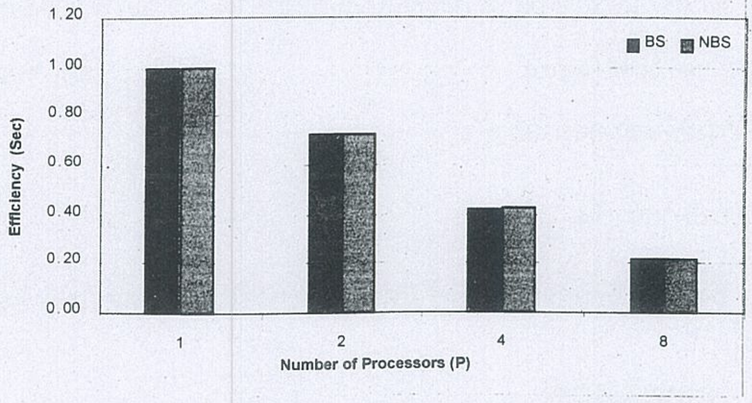

Fig. 8. Efficiency of NBS and BS sorting (N = 50).

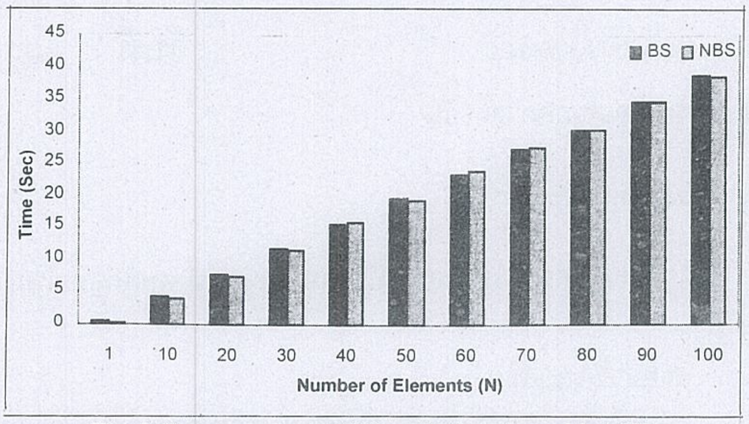

Fig. 9. Response time NBS and BS sorting with 4 PEs

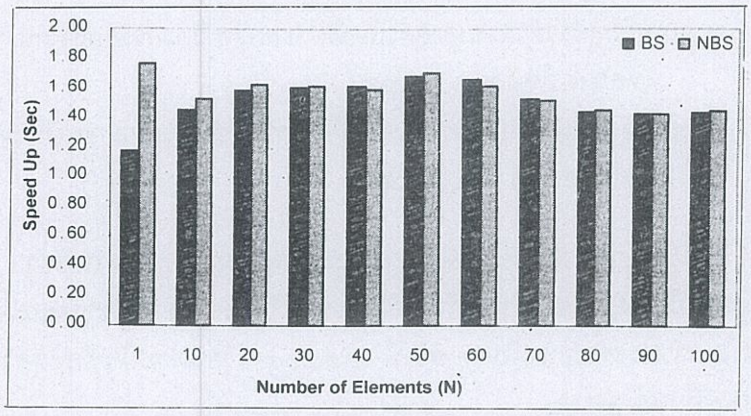

Fig. 10. Speed up of NBS and BS sorting with 4 PEs.

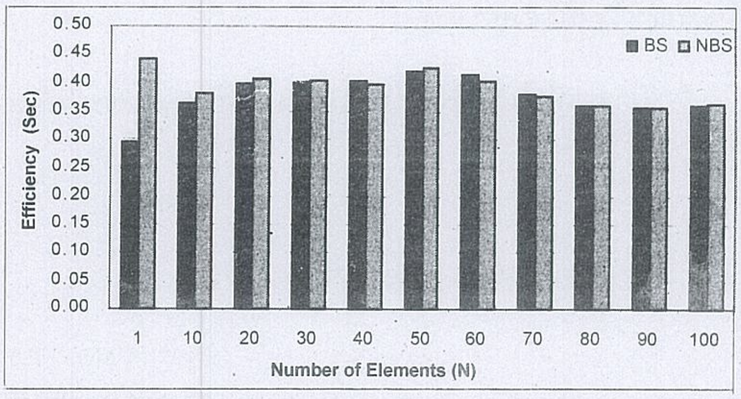

Fig. 11. Efficiency of NBS and BS sorting with 4 PEs.

4. CONCLUSION AND FUTURE STUDY

In this study, we introduce a new space-efficient parallel Bitonic sorting (NBS) on a coarse-grained cluster of PCs (where $P < N$), which can save space up to 50% while computation and communication time is similar to those of the existing parallel Bitonic sorting (BS). The experimental results showed that our approach (NBS) and existing approach (BS) yielded the comparable response time, speedup, and efficiency. In particular, more than one half of the results, our NBS sorting yielded the improved results than those of the BS sorting.

In our future study, we try to improve the communication overhead between processors P_i and P_j by adding the efficient-communication scheme into our space-efficient parallel Bitonic sorting (NBS).

5. ACKNOWLEDGEMENTS

Our work was implemented on the KMITL's cluster of PCs. This cluster system was supported by the computer research and service center at King Mongkut's Institute of Technology Ladkrabang (KMITL), directed by Assoc. Prof. Dr. Manas Sungwarasil and his research assistance, Nopparat Pantsaena.

REFERENCES

[1] Batcher, K.E. **1968** Sorting networks and their applications. *Proceedings Spring Joint Computing Conference AFIPS*, Washington DC, 307-314.

[2] Brest, J. Vreze, A. and Zumer, V. **2000** A Sorting Algorithm on a PC Cluster. *Proceedings 2000 ACM Symposium on Applied Computing (SAC '00)*, Como, Italy, 710-715.

[3] Gropp, W. Lusk, E. and Skjellum, A. **1994** *Using MPI: Portable Parallel Programming with the Massage Passing Interface*. Cambridge, MA, MIT Press.

[4] Helman, D. R. and JaJa, J. **1997** Sorting on Cluster of SMPs. *12th International Parallel Processing Symposium*, University of Maryland, College Park, MD, USA.

[5] Ionescu, M. F. and Schauser, K. E. **1997** Optimizing Parallel Bitonic Sort. *Proceedings 11th Int'l Parallel Processing Symposium*, 303-309.

[6] Kim, Y. C. Jeon, M. Kim, D. and Sohn, A. **2001** Communication-Efficient Bitonic Sort on a Distributed Memory Parallel Computer. *Int' l Conference Parallel and Distributed Systems*, 165-170.

[7] Lee , J. D. and Batcher, K. E. **2000** Minimizing Communication in the Bitonic Sort. *IEEE Transaction on Parallel and Distributed Systems*, 459-473.

[8] Message Passing Interface Forum. **1994** MPI: A message passing interface standard. *Int'l Journal of Supercomputer Applications*, 8(3/4).

[9] www. http://www-unix.mcs.anl.gov/mpi/papers/archive/index.html