공유 메모리 병렬 컴퓨터 환경에서 Bitonic Sorting 알고리즘 설계와 효율적인 통신의 구현

이 재 동† · 권 경 희† · 박 용 범†

요 약

본 연구에서는 공유메모리 병렬 컴퓨터 환경에서 N개의 key를 $O(\log^2 N)$ 시간에 정렬 할 수 있는 병렬 알고리즘인 SHARED-MEMORY-BS와 REDUCED-BS를 설계하였다. REDUCED-BS 알고리즘은 각각 로세서에 있는 local memory를 효율적으로 사용할 수 있도록 제안한 parity전략을 사용하였다. 각각의 프로세서에 있는 local memory를 효율적으로 사용함으로써 REDUCED-BS 알고리즘은 SHARED-MEMORY-BS 알고리즘에 비하여 통신의 빈도수가 약 1/2정도 감소된 것으로 나타났다. 결과적으로 REDUCED-BS 알고리즘은 병렬 정렬시 통신을 감소시킴으로써 컴퓨터의 사용 효율을 향상시킬 수 있다.

Designing a Bitonic Sorting Algorithm for Shared-Memory Parallel Computers and an Efficient Implementation of its Communication

Jae-Dong Lee† · Kyung-Hee Kwon† · Young-B Park†

ABSTRACT

This paper presents parallel sorting algorithms, SHARED-MEMORY-BS and REDUCED-BS, which are implemented on shared-memory parallel computers. These algorithms sort N keys in $O(\log^2 N)$ time. REDUCED-BS uses a parity strategy which gives an idea for the efficient usage of the local memory associated with each processor. By taking advantage of the local memory associated with each processor, the communication of REDUCED-BS is decreased by approximately half that of SHARED-MEMORY-BS. On the basis of alleviating the communication, the algorithm REDUCED-BS results in a significant improvement of performance.

1. Introduction

For both practical and theoretical reasons, sorting is probably the best studied problem in computer science over the past few decades. Sorting is one of the most common operations performed by a computer in both parallel and sequential computing systems. It is often said that twenty-five to fifty percent of all the work performed by computers is being spent on sorting[1]. Parallel sorting allows high performance sorting by utilizing multiple functional units concurrently. Bitonic sort is a popular sorting scheme due to its inherent parallelism [6, 8, 9, 10]. In 1968, the bitonic sorting scheme was introduced to construct a sorting network[2]. Later, the bitonic sorting scheme

※This work was supported in part by Dan-Kook University.
† 정 회 원:단국대학교 전자계산학과
논문접수:1997년 5월 7일, 심사완료:1997년 9월 23일

has been adapted to a variety of parallel computers such as hypercube [7], perfect-shuffle [13], mesh-connected parallel computers [11, 14], and cube-connected-cycles [12].

Although the bitonic sorting scheme has been extensively used in parallel computing (mainly to construct sorting networks), there is a need for a bitonic sort algorithm for a general purpose shared-memory parallel computation model.

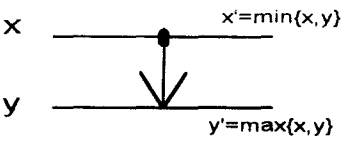
In this paper the bitonic sort algorithm for shared-memory parallel computers is developed and a modified bitonic sorting algorithm is presented, which uses local memories efficiently. The modification improves performance in the sense that the number of shared-memory accesses across the interconnection network is decreased. This improvement is achieved by maintaining and using local memory efficiently.

Section 2 describes basic definitions which are used in this paper. Section 3 describes the parallel models for the considered algorithms. Section 4 introduces the algorithm SHARED-MEMORY-BS developed for shared-memory parallel computers. Section 5 introduces the algorithm REDUCED-BS which reduces the shared-memory accesses. Finally, section 6 states some conclusions and future work.

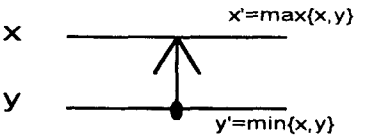
2. Definitions

The key component of the bitonic sorting is a comparison-exchange element (or a *comparator*, for short), as shown in (Fig. 1). Two input keys, x and y, are compared and output in ascending order with $\min(x, y)$ output to x', and $\max(x, y)$ output to y'; for a decreasing comparator $x' = \max\{x, y\}$ and $y' = \min\{x, y\}$. Throughout this paper, the uppercase letter N denotes a power of two, lowercase letter k denotes $\log N$, all logarithms are base two and r' represents the value of 2^r.

An ascending sequence, represented as /-sequence, is a non-decreasing sequence of keys $\{x_0, x_1, x_2, ..., x_{N-1}\}$ such that $x_0 \le x_1 \le ... \le x_{N-1}$. A decending sequence, represented as \-sequence, is a non-increasing sequence of keys $\{x_0, x_1, x_2, ..., x_{N-1}\}$ such that $x_0 \ge x_1 \ge ... \ge x_{N-1}$. Note that a single key, x, is both an /-sequence and a \-sequence.



(a) Increasing comparator



(b) Decreasing comparator

(Fig. 1) A simple 2×2 comparator type.

An *ascending-desceding* sequence (or $\wedge$-sequence) is an /-sequence follow by a \-sequence of key, $\{x_0, x_1, x_2, ..., x_i, x_{i+1}, ..., x_{N-1}\}$ such that $x_0 \le x_1 \le ... \le x_i \ge x_{i+1} \ge ... \ge x_{N-1}$. Note that any /-sequence is an $\wedge$-sequence. Similary, a *desceding-ascending* sequence (or $\vee$-sequence) is a \-sequence followed by an /-sequence.

A *bitonic sequence* is a sequence of keys $\{x_0, x_1, ..., x_{N-1}\}$ with the property that (a) *there exists an index* i, $0 \le i \le N-1$, *such that* $x_0 \le x_1 \le ... \le x_i \ge x_{i+1} \ge ... \ge x_{N-1}$, *or* (b) *there exists a cyclic shift of indices so that* (a) *is satisfied.*

For example, {0 1 2 3 4 5 6} is a bitonic sequence since the sequence {0 1 2 3 4 5 6} may be defined as the /-sequence and {null} as the \-sequence. The sequence {0 1 2 3 4 3 2 1} is also a bitonic sequence, because it first increases and then decreases. Similarly, {3 4 7 6 5 1 0 2} is another bitonic sequence, because it is a cyclic shift of {0 2 3 4 7 6 5 1}. However, the sequence {0 2 3 7 6 4 8 5 1} is not a bitonic sequence since this sequence cannot be an $\wedge$-sequence by the rotation. Notice also that, any sequence of three or less keys is bitonic and any subsequence of a bitonic sequence is also bitonic.

The following theorem gives us the iterative rule to construct the bitonic sorting network [2]. This rule is an essential method to rearrange a bitonic sequence into a sorted sequence in a bitonic sorting.

Theorem 1: Let $X=\{x_0, x_1, ..., x_{N-1}\}$ be bitonic. If $B=\{b_0, b_i, ..., b_{N/2-1}\}$ where $b_i=\min(x_i, x_{N/2+i})$ and $C=\{c_0, ..., c_i, ..., c_{N/2-1}\}$ where $c_i=\max(x_i, x_{N/2+i})$ for $0\le i\le N/2-1$, then (1) both B and C are bitonic and (2) $max\{b_0, b_1, ..., b_{N/2-1}\}\le min\{c_0, c_1, ..., c_{N/2-1}\}$. (See [2])

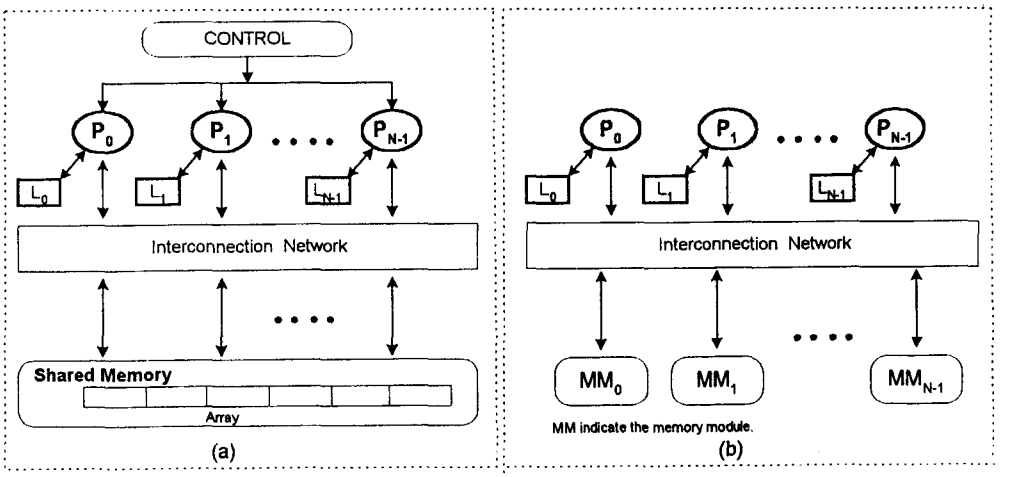
3. Parallel Models of Computation

In this paper, shared-memory SIMD and/or MIMD computers are used in the studying of bitonic sorting algorithms. A Knuth diagram shows how to program the bitonic sorting algorithm using $N/2$ processors for shared-memory parallel computers [5]. (Fig. 2) shows shared-memory SIMD (a) and MIMD (b) computers, where shared-memory could be either centralized as shown in (Fig. 2(a)) or distributed as shown in (Fig. 2(b)).

An advantage of shared-memory SIMD and/or MIMD computers is that they can simulate message-passing SIMD and/or MIMD computers in which the processors communicate through an interconnection network. Therefore, the algorithms considered in this paper can be modified to run on message-passing SIMD and/or MIMD computers while keeping the number of processors and parallel run time requirements unchanged. This modification is straightforward if the direct links connecting the processors are simulated by alternating the write and read operations by the processors, into and from shared-memory, respectively. Throughout this paper, shared-memory parallel computers indicate shared-memory SIMD computers and/or shared-memory MIMD computers.

4. Designing a Bitonic Sorting Algorithm for Shard-memory Parallel Computers

A bitonic sorting algorithm for shard-memory parallel computers (SHARED-MEMORY-BS) is presented in this section. It uses $N/2$ parallel processors to sort a sequence of $N(=2^k)$ keys in $\log N$ stages where stage i requires i steps, for a total of $\log N(\log N+1)/2$ steps. The output of each stage in the algorithm is a concatenation of bitonic sequences that are twice as long as the input. The algorithm for shared-memory parallel computers can be illustrated using a Knuth diagram



(Fig. 2) Shared-memory (a)SIMD and (b)MIMD computers with local memory

[5]. A Knuth diagram shows how to program the algorithm in a parallel processor (see (Fig. 3)). In a Knuth diagram, each horizontal line represents a key in an array (A_i) of N keys and each vertical arrow represents a processor reading two key from the array, comparing them, and writing them back to the array in proper order. Based on the Knuth diagram, the straightforward programming way of the bitonic sorting algorithm for shared-memory parallel computers is briefly described as follows:

/* Each of the $k(k+1)/2$ steps has two parameters, r and d, with $0 \le d \le r \le k$ $(=\log N)$ */
for $r=1$ to k do
for $d=r-1$ down 0 do
/* One step of SHARED-MEMORY-BS with parameters r and d */
1. Each processor reads two keys from an array (*read phase*).
2. Perform a compare-exchange operation (*computation phase*).
3. Each processor writes them back to the array (*write phase*).

A step with parameters r and d forms $N/2$ pairs of keys with two keys in each pair. Let the two keys of a pair be K1 with index bits (b_{k-1}, b_{k-2}, ..., b_1, b_0) and K2 with index bits (c_{k-1}, c_{k-2}, ..., c_1, c_0). Then $c_d=1$ and $b_d=0$; and $c_i=b_i$, for all i except d. The indices of the two keys of each pair differ only in bit d. In general, keys whose indices differ only in bit i perform a compare-exchange operation $(k-i)$ times.

A compare-exchange operation

The key to the algorithm is a compare-exchange operation, which compares the two keys of each pair and swaps if necessary to put them in the proper order. The proper order is determined by parameter r; if $b_r=c_r=0$ then K1 gets the minimum key and K2 gets the maximum key; otherwise, if $b_r=c_r=1$ (case $b_r \neq c_r$ is impossible) then K2 gets the minimum key and K1 gets the maximum key. When $r=k$ then all comparators put the minimum key in K1 and the maximum key in K2.

The following procedures are developed based on the above description. Here, two flags ($RFLAG_{pid}$ and $SFLAG_{pid}$) are used as a way of determining the proper order of two keys. The meaning of these flags is illustrated in (Fig. 3). $RFLAG_{pid}$ determines the direction of the arrow ($RFLAG_{pid}$ = True ⇒ arrow up). $SFLAG_{pid}$ determines the proper order of two keys in the array. PID is the processor ID.

Procedure SET-RFLAG(r)
Global $RFLAG_{pid}$: boolean
1 if $\lfloor 2*PID/2^r \rfloor$ = Odd number
2 then $RFLAG_{pid} \leftarrow$ True
3 else $RFLAG_{pid} \leftarrow$ False

SET-RFLAG determines the value of RFLAG of each processor; each processor sorts two keys ascendingly when $RFLAG_{pid}$ = False (see ⟨Table 1⟩).

The following procedure describes that each processor performs a compare-exchange operation at each step; two keys are compared and exchanged if necessary ($SFLAG_{pid}$ = True ⇒ exchange two keys).

Procedure COMPARE-EXCHANGE
Global $RFLAG_{pid}$, $SFLAG_{pid}$:boolean, $K1_{pid}$, $K2_{pid}$:key
1 Set $SFLAG_{pid} \leftarrow RFLAG_{pid}$.
2 if $K1_{pid} > K2_{pid}$.
3 then Set $SFLAG_{pid} \leftarrow$ Not $SFLAG_{pid}$.
4 if $SFLAG_{pid}$ = True
5 then Swap $K1_{pid}$ and $K2_{pid}$.

Addressing of the array for two keys

In shared-memory parallel computers, each processor reads/writes two keys from/into the shared-memory array in each step. The following procedure obtains the addresses of shared-memory array and returns them to the main procedure SHARED-MEM-

ORY-BS in order to access two keys.

Procedure KEY-ADDRESS (d)
Global Q_{pid}, R_{pid} : integer
1 Set $Q_{pid} \leftarrow \lfloor PID/d \rfloor * d$.
2 Set $R_{pid} \leftarrow PID - Q_{pid}$.
3 Obtain the address of the first key: ADRES1 $\leftarrow 2*Q_{pid} + R_{pid}$.
4 Obtain the address of the second key: ADRES2 $\leftarrow 2*Q_{pid} + R_{pid} + d$.
5 Return (ADRES1, ADRES2).

The following algorithm implements the bitonic sorting algorithm for shared-memory parallel computers, which sorts N keys in $O(\log^2 N)$ time using $N/2$ processors.

Algorithm SHARED-MEMORY-BS
- **Input**: A[0...(N−1)], an array of N keys in shared memory.
- **Output**: Array A with keys in non-decreasing order.
- **Comment**: $N/2$ processors are used. PID is the processor ID ranging from 0 through $N/2$-1. This algorithm runs on shared-memory parallel computers.

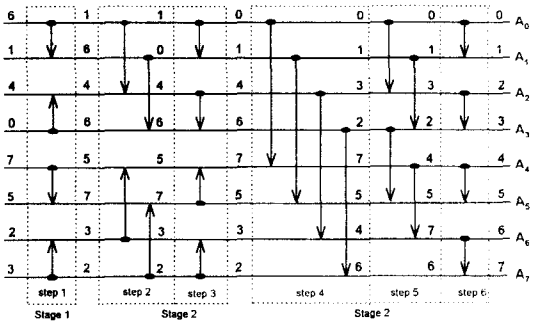
Global $K1_{pid}$, $K2_{pid}$: key $RFLAG_{pid}$: boolean
For all processors in parallel
1 for $r \leftarrow 1$ to k do
2 Set $RFLAG_{pid}$ using the procedure SET-RFLAG(r).
3 $d \leftarrow 2^{r-1}$.
4 While $d \geq 1$ do
5 Obtain two key addresses of an array: (ADR1, ADR2) ← KEY-ADDRESS(d).
6 Read two keys from an array: $K1_{pid}$ ← A[ADR1], $K2_{pid}$ ← A[ADR2].
7 Perform a COMPARE-EXCHANGE for a compare-exchange operation.
8 Write two keys back to the array: A[ADR1] ← $K1_{pid}$, A[ADR2] ← $K2_{pid}$.
9 $d \leftarrow d/2$.

The algorithm first sets $RFLAG_{pid}$ to determine the sorting type of each processor (ascending or descending) in line 2. Within the body of the while loop, each processor reads two keys from the array, performs a

		stage1		stage2				stage3					
		step1		step2		step3		step4		step5		step6	
r		1		2		2		3		3		3	
r'		2		4		4		8		8		8	
d		1		2		1		4		2		1	
Rflag	P0	False		False		False		False		False		False	
	P1	True		False		False		False		False		False	
	P2	False		True		True		False		False		False	
	P3	True		True		True		False		False		False	
Sflag	P0	True		False		True		False		False		False	
	P1	False		True		False		False		False		True	
	P2	True		False		True		True		True		False	
	P3	True		False		False		False		False		True	
Two keys		K1	K2	K1	K2	K1	K2	K1	K2	K1	K2	K1	K2
Key value	P0	6	1	1	4	1	0	0	7	0	3	0	1
	P1	4	0	6	0	4	6	1	5	1	2	3	2
	P2	7	5	5	3	5	7	4	3	7	4	4	5
	P3	2	3	7	2	3	2	6	2	5	6	7	6

Table 1 : The operations of SHARED-MEMORY-BS.

compare-exchange operation and writes two keys back into the array in lines 6-8. (Fig. 3) shows a Knuth diagram for 8 keys, which illustrates how to program the bitonic sorting algorithm for shared-memory parallel computers. 〈Table 1〉 shows how SHARED-MEMORY-BS works using sample values shown in (Fig. 3), where flag conditions are illustrated. Key values which are processed by each processor are shown. By using two flags (Rflag, Sflag), each processor determines the proper order of two keys.



(Fig. 3) A knuth diagram based on the bitonic sort for $N=8$

The following theorem establishes the correctness of SHARED-MEMORY-BS.

Theorem 2: SHARED-MEMORY-BS sorts $N(=2^k)$ keys correctly in O(log2N) time.

Proof: A sequence of two keys K_1 and K_2 forms a bitonic sequence, since either $K_1 \leq K_2$, in which case the bitonic sequence has K_1 and K_2 in the increasing part and no keys in the decreasing part, or $K_1 \geq K_2$, in which case the bitonic sequence has K_1 and K_2 in the decreasing part and no keys in the increasing part. Hence, any unsorted sequence of input keys is a concatenation of bitonic sequence of size two (see stage 1 of (Fig. 3)). After a single compare-exchange step, the pairs can be sorted iteratively ascending and descending. Next all groups of 22 keys are bitonic as a result of the previous step. Two compare-exchange steps are required for sorting the 2^2 keys (see stage 2 of (Fig. 3)). At stage i for $1 \leq i \leq \log N$, $N/2^i$ groups of keys of size 2^i are obtained. During a single step, each processor reads two keys from an array in shared-memory at line 6, compares them if necessary in line 7 and writes them back to the array in line 8. Two flags ($RFLAG_{pid}$ and $SFLAG_{pid}$)are used as a way of determining the proper order of two keys. The same process is applied to groups of size 2^3, 2^4, ..., and $2^{\log N}$. Following $\log N$ stages (i.e., iterations of the for loop), the keys are sorted in an ascending order.

Clearly, the algorithm SHARED-MEMORY-BS takes $\log N(\log N+1)/2$ steps to sort N keys. Therefore, the time complexity of SHARED-MEMORY-BS is $O(\log^2 N)$. □

The following theorem obtains the total processor-memory communications per processor of SHARED-MEMORY-BS.

Theorem 3: Let $C(N)$ be the total number of shared-memory references per processor in SHARED-MEMORY-BS. Then $C(N)=2*k*(k+1)$ for sorting $N(=2^k)$ keys.

Proof: Each step needs four shared-memory references since the *read phase* in line 6 requires two shared-memory references to read and the *write phase* in line 8 requires two shared-memory references to write as well. Thus, the total number of shared-memory references per processor is $C(N)=4*k*(k+1)/2=2*k*(k+1)$ since SHARED-MEMORY-BS requires total $k(k+1)/2$ steps. □

5. Reducing Software Communication

As described in the previous section, the straightforward programming way of the bitonic sorting algorithm for the shared-memory parallel computers is that each processor reads two keys from shared-memory, compares them, and writes two keys back to shared-memory. Thus, four shared-memory accesses are required in each step as described in theorem 3. Since shared-memory access can be very time consuming it is desirable to reduce the number of such

accesses. In the following, an algorithm which reduces the number of shared-memory accesses is introduced.

5.1 The Basic Approach

The basic idea employed in the proposed bitonic sorting is that if each processor has enough local memory to store one key then each processor reads only one key from shared-memory, compares it to the key in its local memory, and writes only one key back to shared-memory. This decreases the number of shared-memory accesses, and hence, leads to a performance improvement. First, we briefly illustrate the *parity strategy* which leads to the modified bitonic sorting algorithm REDUCED-BS.

Parity strategy: *Let the* **parity** *of a key be defined by the number of 1-bits in its index; if the index has an even number of 1-bits then the key has* **even-parity** *(e.g., 0011); if the index has an odd number of 1-bits then the key has* **odd-parity** *(e.g., 1011). Note that in each pairing of the keys during the bitonic sorting each processor compares an even-parity key with an odd-parity key. We can decrease the communication by letting each processor retain the even-parity key in its local memory and just read and write the odd-parity key from and to shared memory.*

5.2 The algorithm REDUCED-BS

In this section, the algorithm REDUCED-BS is presented, which reduces the number of shared-memory references to and from shared-memory by allowing the even-parity keys reside in the local memory associated with each processor.

Assigning even-parity keys to local memory

The procedure LOAD-EVEN first finds the $N/2$ even-parity keys in the shared-memory array, obtains the address of each even-parity key and initially assigns $N/2$ even-parity keys to each local memory. The function FParity takes PID as input and returns the number of 1-bits in its index as output. $KEEP_{pid}$ is the address of an even-parity key in an array and $Keep\_KEY_{pid}$ is used as a local key to keep an even-parity key in local memory (M_{pid}) of each processor.

Procedure LOAD-EVEN
Global $KEEP_{pid}$: integer $Keep-KEY_{pid}$: local key
1 if FParity(PID) = Even number
2 then Set $KEEP_{pid} \leftarrow 2*PID$
3 else Set $KEEP_{pid} \leftarrow 2*PID+1$
4 Load each even-parity key to local memory:
$Keep\_KEY_{pid} \leftarrow A[KEEP_{pid}]$
LOAD-EVEN is illustrated in (Fig. 4-(A))

A compare-exchange operation using local memory

A compare-exchange operation using local memory is more complicated than the one described in the previous section since each processor has to decide which key to keep in its local memory. Three flags ($RFLAG_{pid}$, $PFLAG_{pid}$, and $MFLAG_{pid}$) are utilized as a way of determining whether each processor retains the maximum key or the minimum key in its local memory. The meaning of these flags is illustrated in (Fig. 5). As described in the previous section, $RFLAG_{pid}$ determines the direction of the arrow ($RFLAG_{pid}$ = True $\Rightarrow$ arrow up). $PFLAG_{pid}$ determines which is the greater key index, the local key index (the even-parity key index) or the global key index (the odd-parity key index). Finally, MFLAGpid determines which of the two keys is larger ($MFLAG_{pid}$ = True $\Rightarrow$ local key is larger). RW_{pid} is an index to read an odd-parity key from an array and RW_KEY_{pid} is a key used as an odd-parity key.

Procedure C-EXCHANGE-LOCAL
Global $KEEP_{pid}$, RW_{pid}: integer RW_KEY_{pid},
Keep-KEY_{pid}: key
$PFLAG_{pid}$, $MFLAG_{pid}$, $RFLAG_{pid}$: boolean
1 if $KEEP_{pid} > RW_{pid}$
2 then $PFLAG_{pid} \leftarrow$ True
3 else $PFLAG_{pid} \leftarrow$ False
4 if $Keep\_KEY_{pid} > RW\_KEY_{pid}$
5 then $MFLAG_{pid} \leftarrow$ True

6 else $MFLAG_{pid} \leftarrow$ False
7 if ($RFLAG_{pid} = PFLAG_{pid}$) and $MFLAG_{pid}$
8 then Keep a smaller key in its local memory (M_{pid})
9 else Keep a larger key in its local memory(M_{pid})

The operations of C-EXCHANGE-LOCAL are illustrated in (Fig. 4) and 5. A compare operation determines the value of PFLAG in lines 4-6. An exchange operation is performed depending on the values of three flags in lines 7-9.

Having defined these procedures, the algorithm REDUCED-BS proceeds as follows:

Algorithm REDUCED-BS

- **Input**: A[0...($N-1$)], an array of $N(=2^k)$ keys in shared-memory.
- **Output**: Array A with keys in non-decreasing order.
- **Comment**: $N/2$ processors are used. PID is the processor ID ranging from 0 through $N/2-1$. Each processor has its own local memory (M_{pid}) to store an even-parity key. The symbol $\oplus$ denotes an Exclusive OR operation.

Global $RFLAG_{pid}$: boolean; $KEEP_{pid}$, RW_{pid}: integer; $Keep\_KEY_{pid}$, RW_KEY_{pid}: key
For all processors in parallel.
1 Initialization: Perform the procedure LOAD-EVEN to assign $N/2$ even-parity keys to local memory associated with each processor.
2 for $r \leftarrow 1$ to $\log N(=k)$ do
3 Set $RFLAG_{pid}$ using SET-RFLAG(r).
4 $d \leftarrow 2^{r-1}$.
5 while $d \geq 1$ do
6 Obtain the address of an odd-parity key: $RW_{pid} \leftarrow KEEP_{pid} \oplus d$.
7 Read only an odd-parity key from an array ; $RW\_KEY_{pid} \leftarrow A[RW_{pid}]$.
8 Perform C-EXCHANGE-LOCAL: a compare-exchange operation.
9 Write only one key back to an array: $A[RW_{pid}] \leftarrow RW\_KEY_{pid}$.
10 $d \leftarrow d/2$.
11 Write an even-parity key from local memory to an array: $A[KEEP_{pid}] \leftarrow Keep\_KEY_{pid}$

Loosely speaking, sorting N keys proceeds in $\log N$ stages (i.e., iterations of the **for** loop). At the first stage, the keys are divided into $N/2$ bitonic sequences each of size 2, in which one key (the even-parity key) is in local memory and the other one (the odd-parity key) is in an array. At stage i, where $1 \leq i \leq \log N$, we have $N/2^i$ groups of keys of size 2^i in each group, 2^{i-1} keys reside in individual local memory and 2^{i-1} keys reside in shared-memory array. Those groups are sorted in parallel into bitonic sequences. During a single step in each stage, each processor gets an odd-parity key from an array in lines 6-7, compares it with its local key at line 8. Depending on the order of keys in line 8, such processor writes one key back to an array at line 9 and keeps the other in its local memory. After $\log N$ stages, the keys are sorted in an ascending order.

By comparing SHARED-MEMORY-BS with REDUCED-BS it should be noted that they perform the same comparisons at each step but using different processors. Since the comparisons are performed in parallel, it doesn't matter which processor does which comparison. Hence, it is easily verified that both algorithm yield the same sequence in each step. Therefore, the correctness of REDUCED-BS can be established by its comparison with SHARED-MEMORY-BS(see (Fig. 3) and 5).

5.3 Analysis of the algorithm REDUCED-BS

In this section, the time and communication complexity of REDUCED-BS are analyzed. As shown in

theorem 3, SHARED-MEMORY-BS needs a total $C(N)=2*k*(k+1)$ shared-memory references per processor and its time complexity is $O(\log^2 N)$.

The following theorem establishes that REDUCED-BS decreases the number of shared-memory references by approximately one half compared with SHARED-MEMORY-BS, while preserving the same time complexity.

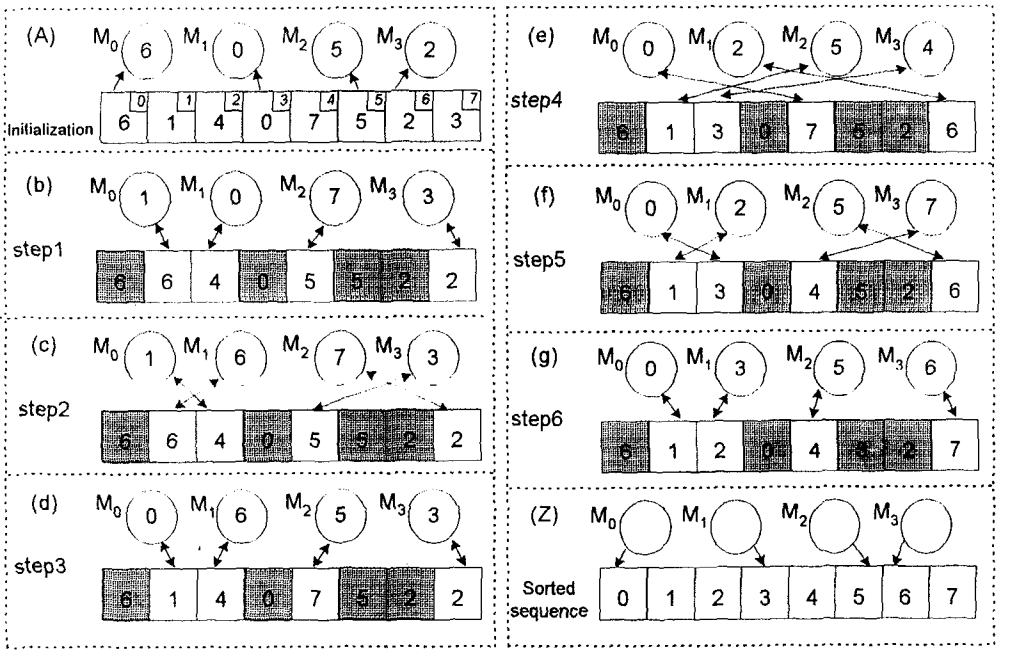
Theorem 4: The total number of shared-memory references per processor of REDUCED-BS is $C(N)/2+2$ and its time complexity is $O(\log^2 N)$ to sort $N(=2^k)$ keys.

Proof: Let $C'(N)$ be the total number of shared-memory references in the algorithm REDUCED-BS. $k(k+1)/2$ steps are required to sort 2^k keys during the execution of the for loop in lines 2-10. Furthermore, each step requires two shared-memory references since one shared-memory access is required to read at line 7 and another shared-memory access is needed to write at line 9. In addition, during the executions of line 1 and line 11, one shared-memory reference is needed, respectively. Therefore, $C'(N)=1+2*k*(k+1)/2+1=k*(k+1)+2=C(N)/2+2$ since $C(N)=2*k*(k+1)$.

The algorithm REDUCED-BS takes O(1) time to initialize at line 1, the execution of the body of the **for** loop takes $O(\log^2 N)$ time, and line 11 requires O(1) time to write the local keys back to the shared-memory array. Therefore, the time complexity of REDUCED-BS $=O(\log^2 N)$. □

5.4 An 8-key REDUCED-BS example

As an example, consider the sorting of 8 keys. The key indices are 3-bit binary numbers, (b_2, b_1, b_0), for 8 keys. (Fig. 4) shows the sorting of an unsorted input list on the array A={6, 1, 4, 0, 7, 5, 2, 3} through REDUCED-BS. In this figure, lightly shared array cell shown the even-parity keys which are never accessed



(Fig. 4) The operations of the algorithm REDUCED-BS for N=8

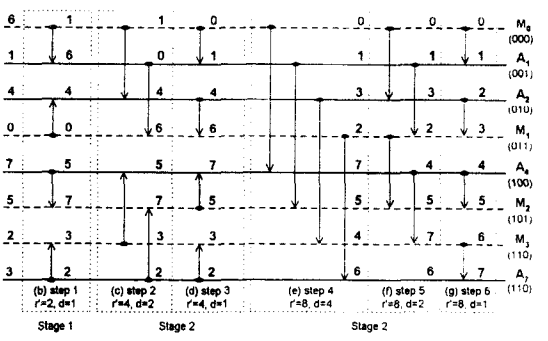
during the execution of the body of the *for* loop. M_{pid} indicates the individual local memory of each processor which keeps an even-parity key, only.

Graph (A) illustrates the initialization (line 1 of REDUCED-BS) with unsorted input keys; every even-parity key is initially assigned to local memory associated with each processor.

Graph (b)-(g) show that the sequence of keys are sorted using local memory which keeps an even-parity key; the execution of the body of the **for** loop of REDUCED-BS. Here, the arrow lines illustrate the compared two keys, an odd-parity key and an even-parity key. During these steps, shared array keys are never accessed.

Graph (Z) illustrates that the local keys are written back into the shared-memory array, so the whole sorted sequence resides in shared-memory; line 11 of REDUCED-BS.

As a further illustration, a Knuth diagram of this example is shown in (Fig. 5). Each solid horizontal line represents an odd-parity key in an array (A_i) and each dashed horizontal line represents an even-parity key in local memory (M_{pid}) associated with each processor. Each vertical arrow represents a processor reading only an odd-parity key from an array, comparing it with an even-parity key in its local memory, and writing one key back to the array in proper order. The indices of each key are shown as a 3-bit representation. Sample values are shown on the wires.



(Fig. 5) A knuth diagram with 4-local memories for $N = 8$

6. Conclusions and Future Work

In this paper, we have presented the bitonic sorting algorithm which are implemented on shared-memory parallel computers and sort N keys in $O(\log^2 N)$ time using $N/2$ processors. The algorithm SHARED-MEMORY-BS which has no memory fetch conflicts is presented. The modified bitonic sorting algorithm REDUCED-BS which is based on SHARED-MEMORY-BS is presented. Using the *parity strategy*, the algorithm REDUCED-BS decreases network communication by taking advantage of local memory associated with each processor. Communication between each processor and shared-memory is reduced by approximately one half compared with the algorithm SHARED-MEMORY-BS. Therefore, the algorithm REDUCED-BS significantly improves performance.

Both algorithms can be adapted for commercially available shared-memory MIMD computers such as a NYU Ultracomputer [3] since memory fetch conflicts are prevented by synchronization. They can be adapted for shared-memory SIMD computers like a BSP machine [4].

Anticipated continued research will find other appropriate applications for the *parity strategy*.

References

[1] Akl, S. G., *Parallel Sorting Algorithms*, Academic Press, Inc., 1985.

[2] Batcher, K.E, "Sorting networks and their applications", *Spring Joint Computer Conference, AFIPS Proc.*, Vol. 32, pp. 307-314, 1968.

[3] Gottlieb, A., Grishman, R., Kruskal, C.P., McAuliffe, K.P., Rudolph, L., and Snir, M., "The NYU Ultracomputer-designing a MIMD, shared memory parallel computer", *IEEE Transactions on Computers*, Vol. C-32, No. 2, pp. 175-189, February 1983.

[4] Hwang, K., *Advanced Computer Architecture: Parallelism, Scalability, Programmability*, McGraw

Hill, Inc., 1993.

[5] Knuth, D., The Art of *Computer Programming: Sorting and Searching*, Vol. 3, Addison-Wesley, 1973.

[6] Kumar, M., and Hirschberg, D., "An Efficient Implementation of Batcher's Odd-Even Merge Algorithm and Its Application in Parallel Sorting Schemes", *IEEE Transactions on Computers*, Vol. C-32, No. 3, pp. 254-264, March 1983.

[7] Kumar, V., Grama, A., Gupta, A. and Karypis, G., *Introduction to Parallel Computing: design and analysis of parallel algorithms*, The Benjamin/Cummings Publishing Company, Inc., 1994.

[8] Kumar, V., Grama, A., Gupta, A and Karypis, G., *Introduction to Parallel Computing: design and analysis of parallel algorithms*, The Benjamin/Cummings Publishing Company, Inc., 1994.

[9] Lee, Jae-dong and Batcher, K.E., "Simplying Multistage Hardware Interconnection in the Bitonic Sorting Network", *in Proceedings of the 7th IASTED/ISMM International Conference on Parallel and Distributed Computing Systems*, pp. 138-142, 1995.

[10] Lee, Jae-dong and Batcher, K.E., "A Bitonic Sorting Network with Simpler Flip-Interconnections", *in Proceedings of the 1996 International Symposium on Parallel Architectures, Algorithms and Networks*, pp. 104-109, 1996.

[11] Nassimi, D. and Sahni, S., "Bitonic sort on a mesh-connected parallel computer", *IEEE Transactions on Computers*, Vol. C-27, No. 1, pp. 2-7, Jan. 1979.

[12] Preparata, F. and Vuillemin, J., "The cube-connected cycles: a versatile network for parallel computation", *Communications of the ACM*, Vol. 24, No. 5, pp. 300-309, 1981.

[13] Stone, H.S., "Parallel processing with the perfect shuffle", *IEEE Transactions on Computers*, Vol. C-20, pp. 153-161, Feb. 1971.

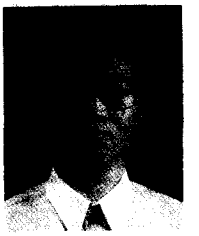
[14] Thompson, C.D. and Kung, H.T., "Sorting on mesh-connected computers", *Communications of the ACM*, Vol. 20, No. 4, pp. 263-271, 1977.

이 재 동

1985년 인하대학교 전자계산학과 (B.S)
1987년~1988년 대우 중공업 정보관리센타
1991년 미국 Cleveland State University, Dept. of computer and Information science (M.S)
1996년 미국 Kent State University, Dept. of computer science (Ph.D)
1992년~1996년 Kent State University, 전산학과 T.A.
1997년~현재 단국대학교 전자계산학과 전임강사
관심분야: 병렬처리, 알고리즘, Interconnection networks, 컴퓨터 네트워크, ATM network

권 경 희

1976년 고려대학교 물리학과 (이학사)
1986년 Old Dominion University, Dept. of Computer Science (M.S)
1991년 Louisiana State University, Dept. of Computer Science (Ph.D)
1979년~1984년 한국산업연구원 연구원
1993년~현재 단국대학교 전자계산학과 조교수
관심분야: 병렬처리, 알고리즘, 연결망



박 용 범

1985년 서강대 전자계산학과 학사 (B.S)
1987년 N.Y. Polytechnic University (M.S)
1991년 N.Y. Polytechnic University (Ph.D)
1992년 현대전자 산전연구소 선임 연구원
1993년~현재 단국대학교 전자계산학과 조교수
관심분야: Speech Recognition, 알고리즘