Bitonic Sort in Shared SIMD Array Processor

Anukul Chandra Panda, Pankaj K Sa
Computer Science and Engineering Department
National Institute of Technology Rourkela
Rourkela, Odisha, India
{pandaa,pankajksa}@nitrkl.ac.in

Banshidhar Majhi*
College of Computer Science
King Khalid University
Abha, Saudi Arabia
bmajhi@nitrkl.ac.in

ABSTRACT

This paper presents a bitonic sort scheme in a shared memory mesh-connected SIMD array processor. In addition, it uses the two types of comparators of sorting networks in the mesh-connected parallel computer. This scheme uses variable multiple pivots and non-pivots. Parity strategy has been implemented to minimize the number of accesses in the mesh-connected interconnection network by introducing the concept of global and local memory. The proposed scheme is sufficiently general which is independent of hardware and interconnection network among them. From results it has been observed that by reducing the internetwork communication a performance improvement is achieved.

Categories and Subject Descriptors

H.4 [**C.1.4**]: Processor ArchitecturesParallel Architectures; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures.*

General Terms

Algorithms.

Keywords

SIMD Array Processor, Bitonic Sort, Parity Strategy.

1. INTRODUCTION

Use of sorted data is an essential ingredient in real life as well as in many computer science applications. This is the reason why the development of efficient sorting algorithms have become a predominant field of computer science research since long. These algorithms not only utilize the computer hardware effectively, but also dynamically change the execution time. Mostly parallel sorting algorithms are implemented in two models, namely special purpose architecture and general purpose architecture. Special purpose architecture includes sorting network and sorting algorithms like AKS algorithm, bitonic sorting algorithms are implemented.

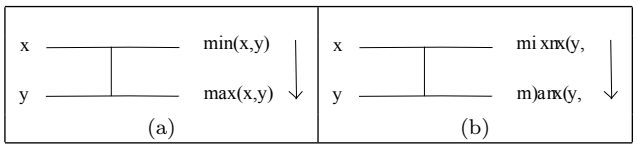

Figure 1: (a) Increasing Comparator (b) Decreasing Comparator

Sorting networks use compare-exchange criteria with sorting complexity $\Omega(n \log n)$ [1] to sort a group of n elements [2, 5]. Two types of comparators i.e. *Increasing Comparator* and *Decreasing Comparator* shown in the Figure 1 are generally used to undergo the sorting using comparison operations in parallel. In this paper, a bitonic sort algorithm on mesh–connected SIMD array processor has been proposed. The proposition uses the comparators of special purpose architecture [3] in general purpose parallel architecture. It uses a shared memory SIMD parallel architecture with mesh connected network topologies. The bitonic sort uses multiple pivots in parallel alongwith a parity strategy [5] to alleviate the spurious communication among processors. However in the proposed scheme multiple pivots have been used. Multiple pivots are used in the proposed bitonic sort on a realistic interconnection network. The base of logarithm is taken as 2 in the paper.

The rest of the paper is organized as follows. Section 2 deals with the proposed parallel bitonic sort scheme in shared memory mesh–connected SIMD parallel architecture. Section 3 gives the time complexity of the proposed scheme. Finally, Section 4 presents the concluding remarks.

2. PROPOSED BITONIC SORT SCHEME

An endeavor has been made to apply bitonic sort using multiple pivots on shared memory SIMD array processor. Bitonic sort algorithm has been implemented in SIMD array processor with static interconnection network i.e. mesh topology [4, 6]. The compare and exchange criterion of sorting network is incorporated on the shared memory mesh-connected SIMD parallel processor. For n elements, there are $\frac{n}{2}$ comparators to sort them in $\log n$ stages. Each comparator associates two processors with itself and performs

*Prof. Dr. Banshidhar Majhi is presently on leave from National Institute of Technology Rourkela, Odisha, India.

ICCCS'11 February 12-14, 2011, Rourkela, Odisha, India

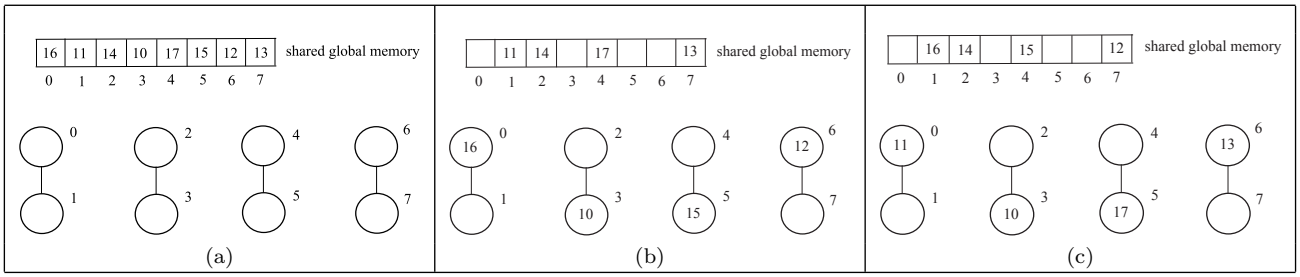

Figure 2: Stage 1 – (a) It shows four (2 × 1) mesh connected processors and a shared global memory, (b) Odd parity elements are kept in the global memory and even parity elements are stored in the local memory, (c) *Step 1*: Shows the state of shared global memory and local memory of processors after the processor-global memory communication as shown in Table 1.

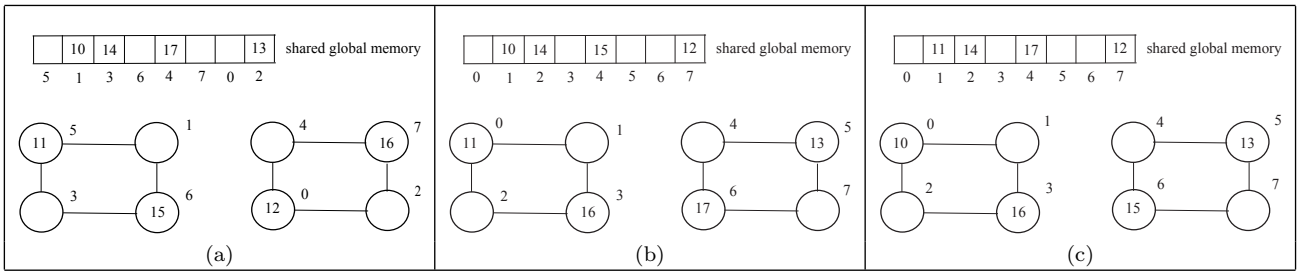

Figure 3: Stage 2 – (a) It shows two (2 × 2) mesh connected processors where odd parity elements are kept in the global memory and even parity elements are stored in the local memory, (b) *Step 1*: Shows the state of shared global memory and local memory of processors after the processor-global memory communication as shown in Table 2, (c) *Step 2*: Shows the state of shared global memory and local memory of processors after the processor-global memory communication as shown in Table 3.

the necessary changes. Each stage j requires j steps for a total of $\frac{\log n(\log n+1)}{2}$ steps.

Here, $\frac{n}{2^j}$ numbers of $(2 \times 2^{j-1})$ (where j = 1, 2, 3, ...) mesh connected parallel processors are used to model each stage. The algorithm to model the mesh connected parallel architecture is done using Knuth diagram [5].

For better understanding of the problem an example has been taken to demonstrate sorting of 8 elements. The elements chosen are 16, 11, 14, 10, 17, 15, 12, 13 stored in shared global memory. Since there are 8 elements, 3 (log 8) stages will be there to perform the sorting of the elements. The figures and tables shown demonstrate the steps required to perform the sorting. Figure 2 through 4 show the inherent steps involved in each stage while Table 1 through 6 depict the mode of communication involved among the processors in the mesh connected SIMD array processor. Figure 2 shows the steps involved in Stage 1. In Figure 2(a), four (2 × 2) mesh connected processors and a shared global memory are taken. In Figure 2(b), elements present in odd parity address are kept in the global memory and the elements in even parity are stored in the local memory of the processor. Figure 2(c) is Step 1 of Stage 1. It shows the state of global memory and local memory of processor after the processor-global memory communication shown in Table 1.

The processors are then taken to form two (2 × 2) mesh connected processors as shown in Figure 3(a). Figure 3 shows steps involved in Stage 2. Figure 3(b) and Figure 3(c) show the states of local memory of processors and global memory after the processor-global memory communication

Table 1: Processor-Global Memory Communication in 2(c)

Comparator	Communication	$FLAG_1$	$FLAG_2$
0	$0 \rightarrow 1$	F	F
1	$2 \rightarrow 3$	T	T
2	$4 \rightarrow 5$	F	T
3	$6 \rightarrow 7$	T	F

Table 2: Processor-Global Memory Communication in 3(b)

Comparator	Communication	$FLAG_1$	$FLAG_2$
0	$0 \rightarrow 2$	F	F
1	$1 \rightarrow 3$	F	T
2	$4 \rightarrow 6$	T	T
3	$5 \rightarrow 7$	T	F

Table 3: Processor-Global Memory Communication in 3(c)

Comparator	Communication	$FLAG_1$	$FLAG_2$
0	$0 \rightarrow 1$	F	F
1	$2 \rightarrow 3$	F	T
2	$4 \rightarrow 5$	T	T
3	$6 \rightarrow 7$	T	F

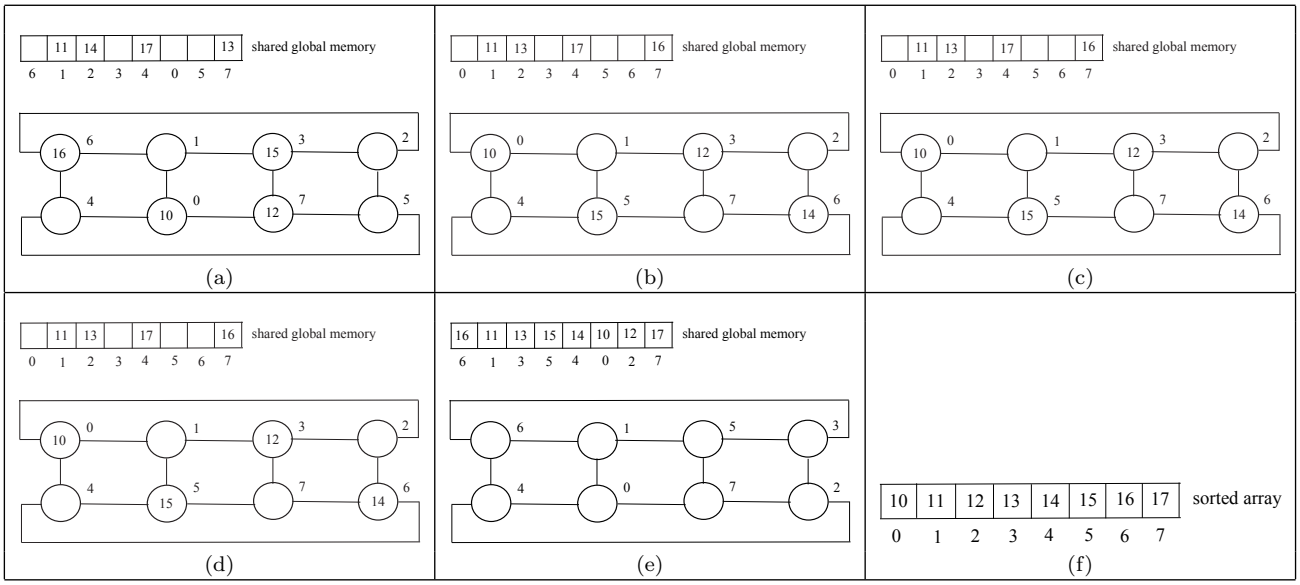

Figure 4: Stage 3 – (a) It shows single (2 × 4) mesh connected processors where odd parity elements are kept in the global memory and even parity elements are stored in the local memory, (b) *Step 1*: Shows the state of shared global memory and local memory of processors after the processor-global memory communication as shown in Table 4, (c) *Step 2*: Shows the state of shared global memory and local memory of processors after the processor-global memory communication as shown in Table 5, (d) *Step 3*: Shows the state of shared global memory and local memory of processors after the processor-global memory communication as shown in Table 6, (e) The elements from the global memory are brought back from the global memory to the respective position into the local memory of the processors, (f) Shows the elements in sorted order.

Table 4: Processor-Global Memory Communication in 4(b)

Comparator	Communication	$FLAG_1$	$FLAG_2$
0	$0 \rightarrow 4$	F	F
1	$1 \rightarrow 5$	F	T
2	$2 \rightarrow 6$	F	T
3	$3 \rightarrow 7$	F	F

Table 5: Processor-Global Memory Communication in 4(c)

Comparator	Communication	$FLAG_1$	$FLAG_2$
0	$0 \rightarrow 2$	F	F
1	$1 \rightarrow 3$	F	T
2	$4 \rightarrow 6$	F	T
3	$5 \rightarrow 7$	F	F

Table 6: Processor-Global Memory Communication in 4(d)

Comparator	Communication	$FLAG_1$	$FLAG_2$
0	$0 \rightarrow 1$	F	F
1	$2 \rightarrow 3$	F	T
2	$4 \rightarrow 5$	F	T
3	$6 \rightarrow 7$	F	F

shown in Table 2 and Table 3, respectively.

Figure 4 depicts Stage 3. In Figure 4(a), single (2 × 4) has been taken. The steps involved to perform the operation from Figure 4(b) through Figure 4(d) is illustrated from Table 4 through Table 6. Figure 4(e) illustrates that the elements are brought back from local memory to global memory of the processor and Figure 4(f) gives the sorted array. The working of $FLAG_1$ and $FLAG_2$ shown in the tables is analyzed through the algorithm given below. The algorithm uses the parity strategy. The main aim behind using the parity strategy is to reduce the interprocessor communication among processors by half. Parity strategy finds the addresses of even parity keys and odd parity keys. $\frac{n}{2}$ multiple pivots is chosen from n elements. Basically, the $\frac{n}{2}$ pivots are chosen from the even parity keys and loaded into local memory. The other $\frac{n}{2}$ elements are chosen as non-pivot elements and stored in the global memory. In this strategy, the pivot and the non-pivot are not fixed since the values present in the even-parity addresses will serve as pivot and vice-versa. So the element keep on changing their role during the course of the $\log n$ stages. Hence, the $\frac{n}{2}$ processors storing non-pivot elements are active since these elements communicate with the local memory to make the necessary changes as depicted in Algorithm 1.

In the algorithm, Line 3 and Line 13 contain two flags ($FLAG_1$ and $FLAG_2$), respectively. $FLAG_1$ is taken to decide the type of comparator described in Section **??** which is to be incorporated within the mesh–connected SIMD array processor. If $FLAG_1$ is TRUE *decreasing comparator* is used otherwise an *increasing comparator* is used. $FLAG_2$ depicts which of the addresses among, local memory (lm)and

Algorithm 1 Bitonic sort with mesh-connected parallel computers

Require: $A[n]$:Array of n elements present in global memory
lm : local memory
no : processor id from 0 to $\frac{n}{2}$-1
gm : global memory
$XOR : \bigoplus$
Ensure: $A[n]$:Array of sorted elements
{*For all processors in parallel*}
1: LOAD $\frac{n}{2}$ elements into lm associated with each processor.
2: LOAD $\frac{n}{2}$ elements into shared gm accessible by all processor.
3: $FLAG_1 \leftarrow$ FALSE
4: **for** j $\leftarrow$ 1 to log n **do**
5: **if** $(\lfloor 2 * no/2^j \rfloor) \bmod 2 \neq 0$ **then**
6: $FLAG_1 \leftarrow$ TRUE
7: **end if**
8: $dim \leftarrow 2^{j-1}$
9: **while** $dim \geq 1$ **do**
10: Obtain the address of a non-pivot element: $gm \leftarrow lm \bigoplus dim$
11: READ only a non-pivot element from odd-parity processor
12: **if** $lm > gm$ **then**
13: $FLAG_2 \leftarrow$ TRUE
14: **else**
15: $FLAG_2 \leftarrow$ FALSE
16: **end if**
17: **if** $FLAG_1 == FLAG_2$ **then**
18: LOAD lowest element into lm
19: **else**
20: LOAD highest element into lm
21: **end if**
22: WRITE only pivot elements into global memory, A
23: $dim \leftarrow \frac{dim}{2}$
24: **end while**
25: **end for**
26: Store all the elements back to array A

global memory (gm) is greater. ($FLAG_2$ = TRUE if address where pivot is stored is greater). In Line 17, $FLAG_1$ and $FLAG_2$ are compared with each other. If $FLAG_1$ and $FLAG_2$ are equal than the lowest element is loaded into the local memory otherwise the highest element is stored . Line 11 demonstrates that only non-pivot element is read from the odd-parity element address. Line 18, 20 and 22 cite that only one element from two elements with each processor is written back to the global memory.

3. TIME COMPLEXITY

In Line 2, a shared memory access is needed. Line 26 requires one more operation. Line 11 and 22 need two memory references. However, since it is within the nested loops, 2 memory references is multiplied by some term which is shown below:

$$\begin{aligned} T(N) &= 2 * (1 + 2 + 3 + \cdots + \log n + 1 + 1) \\ &= 2 * (\log n * (\log n + 1))/2 + 1 + 1 \\ &= \log^2 n + \log n + 2 \end{aligned}$$

So the running time of proposed scheme is $\theta(\log^2 n)$.

4. CONCLUSION

In this paper a scheme has been proposed to apply bitonic sort using parity strategy on mesh-connected SIMD array processor. In the above scheme, sorting network is incorporated in shared memory mesh connected parallel computer to undergo the sorting operation. This approach uses parity strategy which helps in minimizing the shared memory references within the mesh–connected SIMD computers since either odd or even parity elements are allowed to take part in the communication.

5. REFERENCES

[1] M. Ajtai, J. Komlós, and E. Szemerédi. An 0(n log n) sorting network. In *STOC '83: Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pages 1–9, New York, NY, USA, 1983. ACM.

[2] S. Akl. *Parallel Sorting Algorithms*. Academic Press, 1985.

[3] K. E. Batcher. Sorting networks and their applications. In *AFIPS '68 (Spring): Proceedings of the April 30–May 2, 1968, spring joint computer conference*, pages 307–314, New York, NY, USA, 1968. ACM.

[4] K. Hwang. *Advanced Computer Architecture: Parallelism, Scalability,Programmability*. McGraw-Hill, 1993.

[5] Lee, Jae-Dong, Batcher, and K. E. Minimizing communication in the bitonic sort. *IEEE Trans. Parallel Distrib. Syst.*, 11(5):459–474, 2000.

[6] D. Nassimi and S. Sahni. Bitonic sort on a mesh-connected parallel computer. *IEEE Trans. Comput.*, 28(1):2–7, 1979.

[7] P. Sanders and T. Hansch. Efficient massively parallel quicksort. In G. Bilardi, A. Ferreira, R. Lüling, and J. D. P. Rolim, editors, *IRREGULAR*, volume 1253 of *Lecture Notes in Computer Science*, pages 13–24. Springer, 1997.