An AT^2 Optimal Mapping of Sorting onto the Mesh Connected Array without Comparators

Ju-wook Jang*
Dept. of Electronic Eng.
Sogang University
Seoul Korea 121-742
jjang@ccs.sogang.ac.kr

Abstract

In this paper, we present a parallel SIMD algorithm for sorting of N numbers of $\log N$ bits each on a mesh connected array without comparators. While most previous AT^2-optimal sorting algorithms on the mesh explicitly or implicitly assume $O(1)$ time comparison of two operands of $O(\log N)$ bits, our algorithm does not require $O(1)$ time comparison. Rather we assume $O(\log N)$ time comparison, which makes our algorithm realizable with current VLSI technology. To retain the AT^2 optimality with increased (by a factor of $O(\log N)$) comparison time, we develop a new mapping technique which combines radix sort, shear sort, block merge and column sort in a creative way. The time complexity of sorting of N numbers on a (two-dimensional) mesh of size $N^{1/2} \times N^{1/2}$ without comparotors is $O(N^{1/2})$.

1 Introduction

The problem of sorting numbers on a two-dimensional array has been extensively studied and the lower bound is achieved for the mesh connected parallel computer[3, 4, 6, 8] and recently for the reconfigurable mesh[1, 5]. In implementing the sorting problem on a VLSI chip, there exists trade-off between the area (A) of the chip and the processing time (T). It is known that the AT^2-complexity of the sorting N numbers has the lower bound of $\Omega(N^2)$[9]. This implies that sorting of N numbers on a mesh of size $N^{1/2} \times N^{1/2}$ takes $\Omega(N^{1/2})$ time. The algorithms in [3, 6, 1, 5] claim the AT^2-optimality for $T = O(N^{1/2})$ and for $T = O(1)$, respectively. However, there is one common problem for the above algorithms. Their sorting algorithms are based on comparison and they assumed it takes $O(1)$ time to compare two operands in each processing element. Comparison forms such an essential part in sorting that the lower bound of the sorting in the sequential manner is represented sorely by the number of comparisons involved. Most of previously known sorting algorithms on the mesh claims optimality under the implicit or explicit assumption that the comparison of two operands of size $O(\log N)$ can be performed in $O(1)$ time. But the assumption is hardly realizable with current VLSI technology for large N. Thus their algorithms are optimal, but not realizable. If the comparison of two $(\log N)$ bit numbers is implemented in $O(\log N)$ time, the previous algorithms assuming $O(1)$ time comparison become suboptimal by a factor of $\log N$.

In this paper, we assume that the numbers have word size of $\log N$ based on the following observation. If the word size is smaller, the sequential time complexity of the sorting problem would be smaller. Because a radix sort using the bins would result in smaller time complexity. And each PE should know the ID of itself among N PEs. So there should be a storage of size $\Omega(\log N)$ in each PE.

We removed the comparator and instead employed radix sorting of $\log N$ iterations. Sorting N numbers of $\log N$ bits or less can be performed in $\log N$ iterations with the radix sort. In the $i-$th iteration, a stable sort is performed over the N numbers using the $i-$th least significant bit as a key. This replacing of comparators with radix sort has the effect of increasing the time complexity by the factor of $\log N$. To achieve the AT^2-optimality we devised a new mapping scheme to offset the effect and retain the time complexity as low as $N^{1/2}$. The scheme combines radix sort, *shear sort*[7], *block merge*[6] *column sort*[3] and efficient data movement techniques. The numbers are divided into groups

*The author wishes to acknowledge the financial support of the Korea Research Foundation made in the program year of 1997 (No. 1997-001-E00404).

and each group is sorted by radix sort. Merge of the sorted groups is performed by a combination of recursion, shuffle and radix sorting. Initially, the sorting of whole numbers is decomposed into sorting of groups and shuffle between groups. The sorting of each group is again decomposed into sorting of smaller groups and shuffle between them and so on. When the size of the subgroups is small enough, they are sorted by radix sort. Our algorithm is performed in SIMD(Single Instruction stream Multiple Data streams) fashion on mesh connected array of size $N^{1/2} \times N^{1/2}$ with no comparator in each PE(Processing Element). The processing time is $O(N^{1/2})$ time for elements in [0,N] and it satisfies the AT^2-optimality for sorting problem. Compared with previous sorting algorithms[3, 4, 6, 8], our algorithm improves their result by the factor of $O(\log N)$ if it takes $O(\log N)$ time to compare two numbers of $O(\log N)$ bits using registers of size $\log N$).

The rest of the paper is organized as follows: Section 2 briefly describes our parallel computation model and the problem definition. In Section 3, our sorting algorithm is presented. Section 4 concludes this paper.

2 The architecture and problem definition

We will give a brief description of our computation model and the definition of sorting problem.

2.1 The architecture

The mesh connected computer (abbreviated as mesh) is a well-known computation model which has many commercial realizations both in SIMD and MIMD fashion. A mesh of size $M \times N$ denotes a mesh with M rows $\times$ N columns. There is a variety of indexing the PEs. For the purpose of presenting our algorithm, the snake-like row-major ordering is chosen as in Figure 1. The mesh in this paper denotes a weaker model in that it has no comparators and operates in SIMD fashion. This definition of mesh allows simpler implementation of our mesh model on a VLSI chip. Each PE(Processing Element) of an $N \times N$ mesh can store constant number of operands of $(\log N)$ bits. It is assumed that each operand can be moved into one of the PE's four neighboring PEs in $O(1)$ time. This implies the bandwidth between two neighboring PEs is $\Omega(\log N)$. One major difference between our computation model and other related research in the literature is that there is no comparator.It is natural to assume that it takes $O(\log N)$ time to compare two operands of size $\log N$ in our model.

2.2 Problem definition

Initially N elements from a linearly ordered set is stored in the mesh with an element per each PE. After sorting, the elements are placed in such a way that traversing the mesh in snake-like row-major order encounters elements in ascending order. Snake-like row-major order indexing of the mesh is illustrated in Figure 1 with $N = 16$. Transformation among snake-like row-major order and row major order or column major order can be performed in $O(N^{1/2})$ time on a mesh of size $N^{1/2} \times N^{1/2}$. Since the lower bound of any meaningful operations on a mesh has the lower bound of $\Omega(N^{1/2})$ time, the reordering shall not affect the time complexity of the operation.

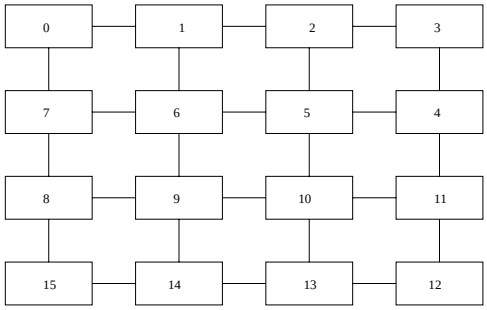

Figure 1. Snake-like row-major ordering of a mesh

3 The sorting algorithm

We provide a skeleton of our algorithm followed by a detailed and formal description of the algorithm using Lemmas and Theorems.

3.1 The skeleton

input N numbers are initially stored on a mesh of size $N^{1/2} \times N^{1/2}$ with each number in each PE(processing element).

output The numbers form an ascending sequence in the snake-like row major order.

processing

1. Divide the mesh into blocks of size $N^{3/8} \times N^{3/8}$.
2. Sort each block. It takes $O(N^{3/8} \log N)$ time using shear sort and radix sort.

3. Perform $N^{1/4}$-shuffle among the blocks. It takes $O(N^{1/2})$ time.
4. Sort each block.
5. Perform inverse $N^{1/4}$-shuffle.
6. Sort each block.
7. Shift the blocks by $N^{1/4}$.
8. Sort each block.
9. Shift the blocks by $-N^{1/4}$.

time complexity Steps 2, 4, 6, and 8 take $O(N^{3/8}\log N)$ time, while steps 3, 5, 7, 9 take $O(N^{1/2})$ time.

3.2 The detailed description

Our algorithm starts with some primitive operations which may have nonoptimal time complexity. The optimality of our algorithm is obtained by combining these nonoptimal operations and known techniques such as radix sort, column sort, shear sort and block merge in a creative way. We also devise new mapping techniques to implement this algorithm on the realizable mesh architecture.

Lemma 1 *It takes $O(N\log N)$ time to sort N numbers in [0, N] on a linear array of size N with no comparators.*

Proof [Radix sort] This can be easily performed by a radix sort:

Do $i = 0$ to $\log N - 1$
    Perform a stable sort on the linear array
    using the i-th bit;
End do
# The 0-th bit is the least significant bit.

Since each stable sort using the i-th bit is performed in $O(N)$ time with no compare operations involved, it takes $O(N\log N)$ time. □

Scherson, Sen and Shamir[7] have shown that an array of size $M \times N$ can be sorted in $\log M$ iterations of sorting the rows in alternating directions and sorting the columns in downward direction (*shear sort*). Combining Lemma 1 and the *shear sort*, we have:

Lemma 2 *It takes $O(N^{1/2}\log^2 N)$ time to sort N numbers in [0, N] on a mesh of size $N^{1/2} \times N^{1/2}$ with no comparators.*

Proof: [Radix sort and Shear Sort]This can be performed as follows:

Do $i = 0$ to $\log N - 1$
    Perform radix sort on all the even-numbered rows from left to right,
    and radix sort on all the odd-numbered rows from right to left;
    Perform radix sort on all the columns;
End do

The radix sort on each row or the radix sort on each column takes $O(N^{1/2}\log N)$ time as in Lemma 1. This is repeated $\log N$ times as dictated in *shear sort*, so it takes $O(N^{1/2}\log^2 N)$ time. □

Now we combine the *block merge* technique[6] to further reduce the time complexity to $O(N^{1/2}\log N)$. Four neighboring sorted blocks are merged into a larger block. Starting from a small block, the merge is recursively performed until the resulting block grows to be the whole mesh. *Zero-one principle*[2] is used to describe the process of *block merge*.

[Zero-one principle] *If a network with N input lines sorts all 2^N sequences of 0s and 1s into nondecreasing order, it will sort any arbitrary sequence of N numbers into nondecreasing order.*

For self-containment, we include a brief description of *block merge* using the zero-one principle. Four blocks of size $k/2 \times k/2$ to be merged into a larger block of size $k \times k$. The small blocks of size $k/2 \times k/2$ are already sorted in snake-like row-major order and in ascending way. Instead of using actual numbers in [0,N], we assume there are only zeros and ones. In the sorted block, we have rows of zeros in the top and rows of ones in the bottom, and there exist at most a row which has mixture of zeros and ones. *Clean row* denotes a row which consists of zeros only or ones only, not both. *Dirty row* denotes a row with mixture of zeros and ones. If there exist at most a dirty row and all the rows above it are clean with zeros and all the rows below it are clean with ones, the block is said to be sorted. And by the zero-one principle, actual numbers can be sorted accordingly. We can add the *block merge* to our algorithm to reduce the time complexity to $O(N^{1/2}\log N)$.

Lemma 3 *It takes $O(N^{1/2}\log N)$ time to sort N numbers in [0, N] on a mesh of size $N^{1/2} \times N^{1/2}$ with no comparators.*

Proof: [Radix sort, Shear Sort and Block Merge]
Without loss of generality, we assume N to be a power of 2. The merge operation is performed in a recursive way. Mergeblock$(0,\ldots,k-1,0,\ldots,k-1)$ merges

four neighboring sorted blocks of size $k/2 \times k/2$ into a sorted block of size $k \times k$. The recursion stops when k becomes $N^{1/2}$:

Mergeblock$(0, \ldots, k-1, 0, \ldots, k-1)$
if $(k \leq \log^2 N)$ then
  sort the block;
else
  Mergeblock$(0, \ldots, k/2-1, 0, \ldots, k/2-1)$;
  Mergeblock$(0, \ldots, k/2-1, k/2 \ldots, k-1)$;
  Mergeblock$(k/2, \ldots, k-1, 0, \ldots, k/2-1)$;
  Mergeblock$(k/2, \ldots, k-1, k/2, \ldots, k-1)$;
Exchange odd-numbered rows of the upperleft
block of size $k/2 \times k/2$ with the
corresponding rows of the upperright block;
Exchange odd-numbered rows of lowerleft
block of size $k/2 \times k/2$
with the corresponding rows of the lowerright block;
Do $i = 0$ to 1
  Perform radix sort on all the even-numbered rows
  from left to right, and radix sort on all
  the odd-numbered rows from right to left;
  Perform radix sort on all the columns;
End do

There exist at most 4 dirty rows in the block of size $k \times k$ after the exchange of rows. Since the number of dirty rows is halved with each iteration of shear sort, it takes two iterations of shear sort to complete the merge of four blocks. Based on these observations, we have:

$$T(k,k) = T(k/2.k/2) + 4k \log N$$

$$T(k/2,k/2) = T(k/4.k/4) + 2k \log N$$

$$T(k/4,k/4) = T(k/8.k/8) + k \log N$$

$$\ldots$$

$$T(\log N, \log N) = O(\log^2 N)$$

The $T(k,k)$ denotes time to sort a block size $k \times k$. From the above equations, we have $T(k,k) = 8k \log N$, $k \geq \log^2 N$ and hence $T(N^{1/2}, N^{1/2}) = O(N^{1/2} \log N)$ follows. □

Finally, we combine the *column sort*[3] with our algorithm to obtain the AT^2 optimality. Leighton[3] showed in *column sort* that N numbers in a matrix of size $r \times s$ can be sorted in column major order in 8 steps, when $r \geq 2(s-1)^2$. In steps 1, 3, 5 and 7, the columns are sorted in downward direction and steps 2 and 4 are transpose and inverse transpose, respectively and steps

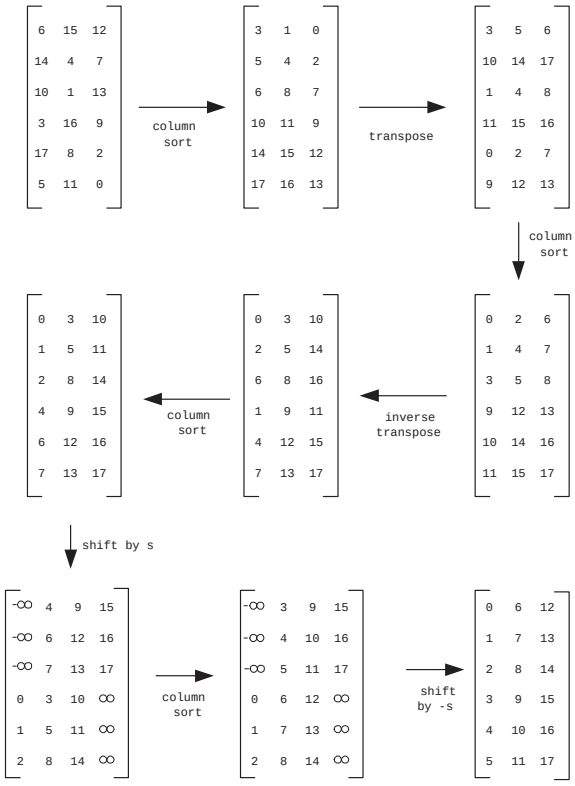

Figure 2. A column sort when $r = 6$ and $s = 3$

6 and 8 are shift by s positions or $-s$ positions, respectively. Figure 2 illustrates the column sort with $r = 6$ and $s = 3$. The detailed proof can be found in [3].

The basic idea is to put our algorithm (the combination of radix sort, *shear sort* and *block merge*) into the 8 steps of the *column sort* and choose r and s in such a way that optimal AT^2 complexity is achieved.

Theorem 3.1 *It takes $O(N^{1/2})$ time to sort N numbers in [0, N] on a mesh of size $N^{1/2} \times N^{1/2}$ with no comparators.*

Proof:
We map the *column sort* onto the mesh as each step of it is being replaced by our combination of the radix sort, *shear sort* and *block merge*. Each column of size r is mapped to a square block of size $\sqrt{r} \times \sqrt{r}$ and transpose and inverse transpose operations are converted into s-shuffle and inverse s-shuffle operations, respectively. The shift operations can be implemented similarly. We choose r and s in such a way that optimal AT^2 complexity is achieved. Since the $A(area)$ is N, T should be no greater than $O(N^{1/2})$. Note that even after combining radix sort, *shear sort* and *block merge* our algorithm is suboptimal by a factor $\log N$(refer to Lemma 3). We choose $r = N^{3/4}$ and $s = N^{1/4}$ and map each column of size r onto a block of size $N^{3/8} \times N^{3/8}$ in the mesh. Then the transpose/inverse transpose operation is converted into $N^{1/4}$-shuffle/inverse $N^{1/4}$-shuffle among these blocks.

Using Lemma 3, steps 1, 3, 5 and 7 can be performed in $O(N^{3/8} \log N)$ time. For steps 2 and 4, there are N numbers to be regularly routed. This can be performed in $O(N^{1/2})$ time since the whole mesh has a bandwidth of $N^{1/2}$ in words of size $\log N$ (i. e. the bisection width of the mesh is $O(N^{1/2})$ words). Steps 6 and 8 can be similarly performed. Thus the overall time complexity is $O(N^{1/2})$ time. □

4 Conclusion

We have presented an AT^2-optimal sorting algorithm on a mesh connected parallel SIMD computer without comparators. Our algorithm replaces the comparison by radix sort and thus is realizable with current VLSI technology. Previous AT^2-optimal sorting algorithms on mesh assumes $O(1)$ time comparison for operands of size $O(\log N)$. Our contribution is in removing this unreasonable assumption and developing new mapping techniques to retain the AT^2-optimality with increased comparison time.

References

[1] J. Jang and V. K. Prasanna, "An Optimal Sorting Algorithm on Reconfigurable Mesh", *Journal of Parallel and Distributed Computing*, Vol. 25, No. 1, pp. 31-41, 1995.

[2] D. E. Knuth, "The Art of Computer Programming: Volume III," *Addison Wesley*, pp. 224-225, 1973.

[3] F. T. Leighton, "Tight bounds on the complexity of parallel sorting," *IEEE Trans. on Computers*, pp. 344-354, 1985.

[4] J. M. Marberg and E. Gafni, "Sorting in Constant Number of Row and Column Phases on a Mesh," *Algorithmica*, Vol. 3, pp. 561-572, 1988.

[5] M. Nigam and S. Sahni, " Sorting n Numbers on $n \times n$ Reconfigurable Meshes with Buses", *Univ. of Florida Dept. of CIS, Tech. report TR-92-04*, 1992.

[6] I. D. Scherson and S. Sen, "Parallel Sorting in Two-dimensional VLSI Models of Computation", *IEEE Transactions on Computers*, C-38, pp. 238-249, 1989.

[7] I. D. Scherson, S. Sen, A. Shamir, "Shear Sort: A True Two-dimensional Sorting Technique for VLSI Networks," *Proc. International Conf. on Parallel Processing*, pp. 903-908, 1986.

[8] C. D. Thomson and H. T. Kung, "Sorting on a Mesh-Connected Parallel Computer," *Communications of the ACM*, Vol. 20, No. 4, pp. 263-271, 1977.

[9] J. D. Ullman, "Computational Aspects of VLSI computation", pp. 48-50 Computer Science Press, 1984.