A FAMILY OF PARALLEL SORTING ALGORITHMS

Yijie Han

Department of Computer Science
Duke University
Durham, NC 27706

Abstract

We generalize Preparata's sorting algorithm into a family of parallel sorting algorithms. This family of sorting algorithms sorts n keys with $O(n\log^\alpha n)$ processors in $O(\frac{1}{\alpha}\log n)$ time units, where α is an arbitrary positive number less than 1. The computation model for this family of algorithms allows simultaneous fetches from the same memory cell.

1 Introduction

Parallel sorting algorithms have been investigated by many researchers and the results obtained demonstrate the tradeoff between time complexities and processor complexities[8]. The computation models employed in designing some of the fast parallel sorting algorithms are classified by Borodin and Hopcroft[10] as PRAC[4], PRAM[5] and WRAM[6]. These are the shared-memory computation models. These models allow each processor access any cell of the shared memory. They differ in the mechanism for resolving the simultaneous read or write of one memory cell by more than one processors. No simultaneous read and write are allowed in the PRAC model. Simultaneous read is allowed in the PRAM model, but not the simultaneous write. Both simultaneous read and write are allowed in the WRAM model. One of the earliest result in parallel sorting is Batcher's bitonic sorting scheme[7] which sorts n keys with n processors in $O(\log^2 n)$[a] time units. Muller and Preparata presented the first parallel sorting algorithm[9] with time complexity $O(\log n)$. But their algorithm requires $O(n^2)$ processors. Valiant studied the parallelism in comparison problems and came up with a sorting algorithm which sorts n keys with n processors in $O(\log n \log\log n)$ time[2]. Hirschberg got a family of parallel sorting algorithms[1] for sorting n keys on PRAM. These algorithms use $O(n^{1+\alpha})$ processors and take $O(\frac{1}{\alpha}\log n)$ time units, where α is an arbitrary positive constant less than 1. Preparata eliminated the fetch conflicts and came up with a family of parallel algorithms[3] for PRAC which have the same time and processor complexities as Hirschberg's algorithms. By using Valiant's fast parallel merging algorithm[2] Preparata derived another parallel sorting algorithm[3] for PRAM which sorts n keys by using $O(n\log n)$ processors and $O(\log n)$ time units. Shiloach and Vishkin studied sorting algorithm on the WRAM model[6].

Recently Ajtai *et. al.* showed n keys can be sorted on a $O(n\log n)$ node network with depth $O(\log n)$[11]. Leighton extended this result and showed n keys can be sorted on a bounded-degree network of $O(n)$ processors in $O(\log n)$ time units[12]. However, the construction of the sorting

*Work reported herein was partially supported by the National Science Foundation under grant MCS - 8202610.

[a]Logarithms in this paper are to the base 2.

network of Ajtai *et.al.* is complicated and the multiplicative constant of the time complexity is huge. The performance will be terrible when $n < 10^{100}$[12].

Simple fan-in argument shows that the lower bound for sorting is $\Omega(\log n)$ on PRAM and PRAC. Consider the class of sorting algorithms for sorting n keys with time complexity $O(\log n)$. Along this line there is a series of results. Muller and Preparata's algorithm uses $O(n^2)$ processors[9]. The family of algorithms for PRAM obtained by Hirschberg has processor complexity $O(n^{1+\alpha})$[1]. The family of algorithm for PRAC obtained by Preparata[3] has the same processor complexity. Preparata's another algorithm for PRAM model has processor complexity $O(n \log n)$[3]. Ajtai *et. al.* and Leighton's results are the optimal results, although their construction is complicated and the multiplicative constant is prohibitively huge.

In this paper we generalize Preparata's algorithm into a family of algorithms for PRAM which uses $\lfloor n \log^{\alpha} n \rfloor$ processors and take $O(\frac{1}{\alpha} \log n)$ time units, where α is an arbitrary positive number less than 1. The multiplicative constant in the timing is estimated to be less than 5.

2 The Family of Parallel Sorting Algorithms

We now present the family of parallel sorting algorithms which sort n keys in time $O(\frac{1}{\alpha} \log n)$ with only $\lfloor n \log^{\alpha} n \rfloor$ processors. The family of algorithms is a generalization of Preparata's result(with $O(n \log n)$ processors). The scheme is basically the same as Preparata's scheme with the exceptions that the keys are divided into about $\lceil \log^{\alpha} n \rceil$ blocks instead of $\lceil \log n \rceil$ blocks and that when n is less than a constant $C(\alpha)$ (which depends on α), Valiant's parallel sorting algorithm[2] (or any other sorting algorithm which sorts n keys with n processors) is employed. The algorithm is as follows.

SORT(A[1:n])

1. If $n < C(\alpha)$, sort the sequence with Valiant's sorting algorithm.
2. Otherwise divide A into blocks such that each block contains $\lfloor n / \lceil \log^{\alpha} n \rceil \rfloor$ keys, with the last block containing less than $\lceil \log^{\alpha} n \rceil$ keys.
3. Recursively SORT all blocks in parallel.
4. Make $\lceil \log^{\alpha} n \rceil$ copies of each block. Thus for every pair of blocks i and $j, (i < j)$, we have a distinct copy of block i and block j.
5. Merge every pair of blocks $i, j (i < j)$ in parallel by using Valiant's fast merging algorithm. Since enough copies were created in the last step, the merging can be done in parallel. After merging each key knows its rank in every block.
6. For each key sum the ranks over all the blocks to get the global rank.
7. Route each element to its destination by using its global rank.

Now we analyze the time and processor complexities.

For processor complexity, steps 4, 5, 6 each use $\frac{n \lceil \log^{\alpha} n \rceil}{2}$ processors. Step 7 uses n processors. The key point for analyzing the complexity is the number of processors used in step 3. The number of processors used in this step is,

$\lceil \log^\alpha n \rceil \lfloor \lfloor n/\lceil \log^\alpha n \rceil \rfloor \log^\alpha \lfloor n/\lceil \log^\alpha n \rceil \rfloor \rfloor$
$+\lfloor \lceil \log^\alpha n \rceil \log^\alpha \lceil \log^\alpha n \rceil \rfloor$
$\le n \log^\alpha (n/\lceil \log^\alpha n \rceil) + \lceil \log^\alpha n \rceil \log^\alpha \lceil \log^\alpha n \rceil$
$= n \log^\alpha n (1 - \frac{\log \lceil \log^\alpha n \rceil}{\log n})^\alpha + \lceil \log^\alpha n \rceil \log^\alpha \lceil \log^\alpha n \rceil$
$\le n \log^\alpha n (1 - \alpha \frac{\log \lceil \log^\alpha n \rceil}{\log n}) + \lceil \log^\alpha n \rceil \log^\alpha \lceil \log^\alpha n \rceil$
$= n \log^\alpha n - (\frac{\alpha n \log \lceil \log^\alpha n \rceil}{\log^{1-\alpha} n} - \lceil \log^\alpha n \rceil \log^\alpha \lceil \log^\alpha n \rceil)$
$\le n \log^\alpha n - 1$, when n is sufficiently large,
$\le \lfloor n \log^\alpha n \rfloor$.

For each value α, there is a constant $C(\alpha)$, when $n \ge C(\alpha)$ the number of processors is enough for another recursive call to SORT, otherwise we'll use Valiant's sorting algorithm to sort n keys with n processors. Although the equation relating C and α might be complicated, the constant C can be calculated in advance. Below we give an estimate. We show that when $n \ge (\frac{2}{\alpha})^4$, the above inequalities hold.

We should find $C(\alpha)$ such that when $n \ge C(\alpha)$,
$\frac{\alpha n \log \lceil \log^\alpha n \rceil}{\log^{1-\alpha} n} - \lceil \log^\alpha n \rceil \log^\alpha \lceil \log^\alpha n \rceil \ge 1.$

Since

$\frac{\alpha n \log \lceil \log^\alpha n \rceil}{\log^{1-\alpha} n} - \lceil \log^\alpha n \rceil \log^\alpha \lceil \log^\alpha n \rceil$
$= \lceil \log^\alpha n \rceil \log^\alpha \lceil \log^\alpha n \rceil (\frac{\log^\alpha n}{\lceil \log^\alpha n \rceil} \frac{\alpha n \log^{1-\alpha} \lceil \log^\alpha n \rceil}{\log n} - 1)$.
When $n \ge (\frac{2}{\alpha})^4$, $\lceil \log^\alpha n \rceil > 1$, $\log^\alpha \lceil \log^\alpha n \rceil \ge \log^\alpha \log^\alpha n \ge \alpha^\alpha \log^\alpha \log n > \alpha^\alpha > \alpha$, $\frac{\log^\alpha n}{\lceil \log^\alpha n \rceil} \ge \frac{1}{2}$, and $\log^{1-\alpha} \lceil \log^\alpha n \rceil \ge \alpha^{1-\alpha} \log^{1-\alpha} \log n \ge \alpha^{1-\alpha} \ge \alpha$. Thus the only thing we need to do is to check that $\alpha(\frac{\alpha^2 n}{2 \log n} - 1) > 1$. When $n \ge (\frac{2}{\alpha})^4$, $\frac{\alpha^2 n}{2 \log n} - 1 \ge \frac{2}{\alpha^2 \log \frac{2}{\alpha}} - 1 \ge \frac{2}{\alpha} - 1 \ge \frac{1}{\alpha}$. Thus we can let $C(\alpha) = (\frac{2}{\alpha})^4$.

For timing, step 1, 2, 4 , 7 each takes constant time. Step 5 takes $C_1 \log \log n$ time. Step 6 takes $\alpha \log \log n$ time. Let $T(n)$ denote the time complexity for sorting n keys, we have the following recursive formula.
$T(n) = T(\frac{n}{\log^\alpha n}) + (C_1 + \alpha) \log \log n + C_2$
for some constants C_1 and C_2. By induction one can show
$T(n) = O(\frac{1}{\alpha} \log n)$.
The induction base can be chosen as $n = \lceil (\frac{2}{\alpha})^4 \rceil$. For $n < C(\alpha)$ the timing will be $O(\log n \log \log n)$ by virtue of Valiant's algorithm. The time complexity $O(\frac{1}{\alpha} \log n)$ will be applicable as long as $O(\log n \log \log n) \le O(\frac{1}{\alpha} \log n)$. This yields $n \le 2^{2^{\frac{d}{\alpha}}}$ for some constant d. By choosing an appropriate constant $d (d < 5)$, we'll have $C(\alpha) \le 2^{2^{\frac{d}{\alpha}}}$. Thus the time complexity $O(\frac{1}{\alpha} \log n)$ will apply to all n.

As suggested by R.A. Wagner and A. Wigderson, the time-processor product is minimized when $\alpha = \dfrac{1}{\ln \log n}$. This yields the result of sorting n keys with $O(n)$ processors in $O(\log n \log\log n)$ time. This result matches the performance of Valiant's sorting algorithm within a constant factor.

3 Conclusions

The family of algorithms presented in this paper generalizes Preparata's algorithm. The recursive structure of the sorting algorithms is simple and easily understood. Although the result presented here is not optimal, it may still provide insights for designing efficient and simple structured parallel sorting algorithms.

Acknowledgments

I wish to offer my thanks to my advisor Dr. Robert Wagner for his encouragement and comments on this work. I appreciate Wagner and Avi Wigderson's suggestion of choosing the optimal α to minimize the time-processor product. The help provided by Dr. Reuven Bar-Yehuda is also appreciated.

References

[1] D.S. Hirschberg. Fast parallel sorting algorithms, Comm. ACM, 21, 8(Aug. 1978), pp. 657-661.

[2] L.G. Valiant. Parallelism in comparison problems, SIAM J. Comput. 4, 3(Sept. 1975), pp. 348-355.

[3] F.P. Preparata. New parallel-sorting schemes, IEEE Tran. Comput. 27, 7(July, 1978), pp. 669-673.

[4] G.F. Lev, N. Pippenger and L.G. Valiant. A fast parallel algorithm for routing in permutation networks, IEEE Tran. Comput. 30, 2(Feb. 1981), pp. 93-100.

[5] S. Fortune and J. Willie. Parallelism in random access machines, Proc. 10th ACM Symposium on Theory of Computing, San Diego, California(1978), pp. 114-118.

[6] Y. Shiloach and U. Vishkin. Finding the maximum, merging, and sorting in a parallel computation model, J. Algorithms 2(1981), pp. 88-102.

[7] K.E. Batcher. Sorting networks and their applications, Proc. AFIPS Spring Joint Computer Conf., vol. 32. Apr. 1968, pp. 307-314.

[8] D. Bitton, D.J. Dewitt, D.K. Hsaio, J. Menon. A Taxonomy of Parallel Sorting, ACM Computing Survey, 16, 3(Sept. 1984), pp. 287-318.

[9] D.E. Muller and F.P. Preparata. Bounds to complexities of networks for sorting and for switching, J. ACM 22, 2(April 1975), pp. 195-201.

[10] A. Borodin and J.E. Hopcroft. Routing, merging and sorting on parallel models of computation, Proc. 14th ACM Symposium on Theory of Computing, San Francisco, April 1982, pp. 338-344.

[11] M. Ajtai, J. Komlos, and E. Szemeredi. An O(nlogn) sorting network, Proc. 15th ACM Symposium on Theory of Computing, Boston, MA, Apr. 1983, pp. 1-9.

[12] T. Leighton. Tight bounds on the complexity of parallel sorting, IEEE Tran. Comput. 34, 4(April, 1985), pp. 344-354.