Tight Comparison Bounds On The Complexity Of Parallel Sorting

by

Yossi Azar[1]
Uzi Vishkin[2]

Ultracomputer Research Laboratory
Courant Institute of Mathematical Sciences
715 Broadway, 10th Floor
New York, NY 10003

Ultracomputer Note #103
Computer Science Department Technical Report #223
February, 1986

[1] Department of Computer Science, School of Mathematical Sciences, Tel Aviv University.

[2] Department of Computer Science, Courant Institute of Mathematical Sciences, New York University and (present address) Department of Computer Science, School of Mathematical Sciences, Tel Aviv University. The research of this author was supported by NSF grant NSF-DCR-8318874 and ONR grant N00014-85-K-0046.

ABSTRACT

The problem of sorting n elements using p processors in a parallel comparison model is considered. Lower and upper bounds which imply that for $p \geq n$, the time complexity of this problem is $\Theta(\frac{logn}{log(1+p/n)})$ are presented. This complements [AKS-83] in settling the problem since the AKS sorting network established that for $p \leq n$ the time complexity is $\Theta(\frac{n\log n}{p})$. To prove the lower bounds we show that to achieve $k \leq \log n$ parallel time, we need $\Omega(n^{1+1/k})$ processors.

1. Introduction

Apparently, there is no problem in Computer Science which received more attention than sorting. [Kn-73], for instance, found that existing computers devote approximately a quarter of their time to sorting. The advent of parallel computers stimulated intensive research of the sorting with respect to various models of parallel computation. Extensive lists of references which recorded this activity are given in [Ak-85], [BHe-86] and [Th-83].

Most of the fastest serial and parallel sorting algorithms are based on binary comparisons. In these algorithms the number of comparisons is typically the primary measure of time complexity. Any lower bound on the number of comparisons required for a problem, clearly implies a time lower bound for such algorithms. In the present paper, we restrict our attention to a parallel comparison model where only comparisons are counted. In measuring time complexity within this model, we do not count steps in which communication among the processors, movement of data and memory addressing are performed. We also avoid counting steps in which consequences are deduced from comparisons that were performed. Note that our lower bounds apply to all algorithms, based on comparison, in any parallel access machine (PRAM) including PRAMs which allow simultaneous access to the same common memory location for read and write purposes. See [BHo-82] for a discussion on hierarchy of models that implies this.

In a serial decision tree model, we wish to minimize the number of comparisons. The goal of an algorithm in a parallel comparison model is to minimize the number of comparison rounds as well as the total number of comparisons performed.

Let k stand for the number of comparison rounds (time) of an algorithm in the parallel comparison model. Given an algorithm, let $u(k,n)$ denote the *total* (over all rounds of the algorithm) number of comparisons required by the algorithm to sort any n elements in k rounds. $u(k,n)$ is the upper bound on the number of comparisons in the worst case. Let $c(k,n)$ denote the *minimum total* number of comparisons required to sort any n elements in k rounds (over all possible algorithms).

The known $\Omega(n\log n)$ comparisons lower bound for sorting in a serial decision tree model implies that, for any k, $c(k,n) = \Omega(n\log n)$. This lower bound can be matched by upper bounds as follows: For $k = c\log n$, the sorting network of [AKS-83] obtains $u(k,n) = O(n\log n)$, where $c > 0$ is a constant which is implied by the network. For $k > c\log n$, the result $u(k,n) = O(n\log n)$ also holds. To see this, simply simulate the AKS network by slowing it down to work in k rounds.

For $k = 1$, $c(1,n)=½(n^2-n)$. This is since any sorting algorithm which works in one round must perform all comparisons. Otherwise, suppose that a dispensed comparison is between two successive elements in the sorted order; the algorithm will clearly fail to distinguish their order. On the other hand, observe that performing all comparisons simultaneously yields an one round algorithm in the parallel comparison model that matches exactly this lower bound, i.e., $u(1,n)=½(n^2-n)$.

So, it remains to consider the situation for $1 < k \le c\log n$.

We state our **main result** which is proved in Section 3:

$c(k,n) > k(\frac{n^{1+\frac{1}{k}}}{e} - n)$ for any $k,n \ge 1$, where e is the base of the natural logarithm.

Corollaries of the main result:

Suppose we have p processors with the interpretation that each processor can perform at most one comparison at each round. Observed that $kp \ge c(k,n)$ or $p \ge c(k,n)/k$. Therefore,

Corollary 1. Any k-round ($k \ge 1$) parallel algorithm for sorting n elements needs $p > \frac{n^{1+\frac{1}{k}}}{e} - n$ processors. This yields $p = \Omega(n^{1+\frac{1}{k}})$ for $k \le clogn$ where[3] c is any constant such that $0 < c < 1$.

Corollary 2. The number of rounds required to sort n elements using $p \ge n$ processors is $k = \Omega(\frac{\log n}{\log(1 + \frac{p}{n})})$.

Proof: $p > \frac{n^{1+\frac{1}{k}}}{e} - n$ implies $1+\frac{p}{n} > \frac{n^{\frac{1}{k}}}{e}$ and therefore $k > \frac{\log n}{1+log(1+\frac{p}{n})}$. Hence, for $p \ge n$, $k = \Omega(\frac{\log n}{\log(1+\frac{p}{n})})$.

Corollary 3. If $p = n\log^{\beta} n$ for $\beta > 0$ then the number of rounds required to sort n elements is $k = \Omega(\frac{logn}{\beta loglogn})$. This is an immediate corollary of Corollary 2.

A parallel algorithm is said to achieve *optimal speed up* if its running time is proportional to $\frac{Seq(n)}{p}$, where $Seq(n)$ is a lower bound on the serial running time, n is the size of the problem being considered and p is the number of processors used.

Corollary 4. If the number of processors is larger than n by an order of magnitude then it is impossible to design an optimal speed up comparison sorting algorithm. More formally, suppose that the number of

[3]Throughout the paper, log refers to the natural logarithm.

processors p is not $O(n)$ (i.e., $n = o(p)$) then there is no (comparison) sorting algorithm which runs in time $O(\frac{n\log n}{p})$.

Section 4 presents upper bounds which match these new lower bounds. Specifically, we describe a parallel comparison algorithm that sorts n elements in $O(\frac{\log n}{\log(1 + \frac{p}{n})})$ rounds using $p \geq n$ processors.

To understand better the significance of the lower and upper bounds of the present paper we will use one more equivalent formulation of the results.

Corollary 5. Suppose we are given $p \geq n$ processors to sort n elements. The total number of comparisons performed by the fastest parallel sorting algorithm is

$$\Theta\left[\frac{p/n}{log(1+p/n)}\ nlogn\right].$$

The factor $n\log n$ represents the serial lower and upper bounds for sorting using comparisons. The other factor represents the deviation from optimal speed up.

In studying the limit of parallel algorithms it is interesting to identify asymptotically the minimal time k that can be achieved by an optimal speed up algorithm. We call this minimal time the parallelism *break point* of the problem being considered. [Va-75] proved that $\Theta(log\log n)$ is the break point for finding the maximum among n elements. [BHo-82] gave a lower bound and [Kr-83] an upper bound to prove that $\Theta(log\log n)$ is the break point for merging two sorted lists, where n is the length of each list. The above two lower bounds were also obtained in a parallel comparison model (which is therefore often referred to as Valiant's model). The present paper enables to add sorting to the list of problems for which the break point was identified. Specifically, Corollary 4 complements the sorting network of [AKS-83] in proving that $\Theta(logn)$ is the break point for sorting n elements. It is interesting to compare the "pattern" in which the break point occurs in these three problems. The elegant lower bound proofs of Valiant and Borodin-Hopcroft show that $\Omega(log\log n)$ rounds are required if n processors are used for the problems of finding the maximum and merging, respectively. The algorithms of Valiant and Kruskal run in $O(log\log n)$ rounds using $\frac{n}{\log\log n}$ processors for each of these problems, respectively. This isolates distinctly the break points for these two problems since the asymptotic time bound can not improve by increasing the number of processors from $\frac{n}{\log\log n}$ to n. On the other hand, such degenerate isolation does not occur in the sorting problem. Specifically, Corollary 5 implies that increasing the number of processors asymptotically always yield asymptotic decrease in the number of comparison rounds.

We note that we have proved a first non trivial lower bound in a parallel comparison model for $\log n$ time. The problem solved in this paper was open for sometime. Interestingly, our proof is relatively simple and based only on elementary methods from discrete and continuous mathematics.

Let us review works on sorting n elements in a parallel comparison model. Haggkvist and Hell [HH-81] proved that if k, the number of rounds, is *constant* , then $\Omega(n^{1+1/k})$ processors are required to sort n elements. Using random graphs and a certain probability space, Bollobas and Thomason [BT-83] proved that almost every algorithm that uses $p = O(n^{3/2} \log n)$ processors sorts n elements in two rounds. Bollobas [Bo-86] iterated the random graphs techniques to prove that n elements can be sorted in a *constant* number of rounds k using $O(n^{1+1/k} \log n)$ comparisons. This almost matches the Haggkvist-Hell lower bound. *Remark.* Conversely, these results imply that for $p = O(n^{1+\varepsilon})$ processors, it is impossible to sort in less than $k = 1/\varepsilon$ rounds, but we can sort in $k = 1/\varepsilon + 1$ rounds. So these upper and lower bounds are at most one round apart when k is constant.

However, a closer look at this lower bound of Haggkvist and Hell reveals the following. They actually proved that if k, the number of rounds, is a variable, then $p > \frac{n^{1+1/k}}{2^{k+1}k} - \frac{n}{2k}$ processors are required to sort n elements. We compare this result with Corollary 1 which was given above. Observe, that their proof implies that $p = \Omega(n^{1+1/k})$ *only* when k is constant. Even for constant k which is not very large this asymptotic lower bound contains a very small constant factor (for instance, if k=100 then $n^{1+1/k}$ is multiplied by less than 2^{-100}). Moreover, their result becomes trivial for $k \geq \sqrt{\log n}$. This is since for this range their result implies an asymptotic bound which is $O(n)$ for the number of processors p as can be readily verified. On the other hand, Corollary 1 states that $p > n^{1+1/k}/e - n$ for every k. As was indicated above, this implies that $p = \Omega(n^{1+1/k})$, for any $k < c \log n$, where $0 < c < 1$ is a constant.

We note two additional papers whose titles are related to the title of the present paper. [Le-84] proposed an adaptation of AKS network to bounded degree n-node networks. [MW-85] gave a $\sqrt{\log n}$ lower bound for parallel sorting by n processors in some variant of PRAM. Their model is not comparable to the parallel comparison model considered here. The trivial $\log n$ lower bound for parallel sorting by n processors in the parallel comparison model does not allow non comparison algorithms like bucket sort. On the other hand, ranking an element among n other elements can be done in one round of comparisons using n processors in the parallel comparison model, while their PRAM seems to require non constant time using n processors.

2. The Parallel Comparison Model

Let V be a set of n elements taken from a totally ordered domain. The *parallel comparison model of computation* allows algorithms that work as follows. The algorithm consists of time steps called *rounds*. In each round binary comparisons are performed simultaneously. The input for each comparison are two elements of V. The output of each comparison is one of the following two: $<$ or $>$. Note that we do not allow equality between two elements of V. This can be done without loss of generality, since we define the order between two equal input elements to be the order of their indices. Each item may take part in several comparisons during the same round.

Remark. Our discussion uses the following correspondence between each round and a graph. The elements are the vertices. Each comparison to be performed is an undirected edge which connects its input

elements. Each computation results in orienting this edge from the largest element to the smallest. Suppose we performed r rounds where $r > 0$ is some integer. Consider any function of V that can be computed using the comparisons performed in these r rounds without any further comparisons of elements in V. Our model defines such a function to be *computable following round r*. Note that this definition suppresses all computational steps that do not involve comparisons of elements in V. Which comparisons to perform at round $r+1$ and the input for each such comparison should be functions which are computable following round r. We are interested in sorting the elements in V from the smallest to the largest in k rounds, where the integer k can be either constant or a function of n.

Recall that c(k,n) denotes the minimum *total* number of comparisons required to sort any n elements in k rounds (over all possible algorithms).

3. The Lower Bound

The Main Theorem: $c(k,n) > k(\frac{n^{1+\frac{1}{k}}}{e} - n)$ for any $k,n \geq 1$, where e is the base of the natural logarithm.

Proof of the Main Theorem: By induction on k and n.

The base of the induction. For k=1 and every $n \geq 1$, clearly $c(1,n) = \frac{n^2-n}{2} > \frac{n^2}{e} - n$. For $n = 1,2$ and every $k \geq 1$, $c(k,1) \geq 0 > k(\frac{1}{e} - 1)$, $c(k,2) > 0 > k(\frac{4}{e} - 2) \geq k(\frac{2^{1+1/k}}{e} - 2)$.

The inductive assumption: Given k,n, if $k' \leq k$ and $n' < n$, or $k' < k$ and $n' \leq n$, then $c(k',n') > k'(\frac{n'^{1+\frac{1}{k'}}}{e} - n')$.

Take any k-round algorithm for sorting a set V of n elements. The first round of the algorithm consists of some set E of comparisons. Recall that we look at them as edges in the graph $G = (V,E)$. An *independent set* in G is a subset of vertices from V such that no two vertices are adjacent by an edge in E. An independent set is *maximal* if it is not a proper subset of another independent set. Consider the graph of the first round of comparisons. Let S be a maximal independent set in this graph and denote $x = |S|$ Each of the $n-x$ elements of $\overline{S}$ must share an edge with an element of S, otherwise S is not maximal. For our lower bound proof, we restrict our attention to linear orders on V, in which each element of S is greater than each element of $\overline{S}$. For any of these orders it is impossible to obtain any information regarding the relation between two elements of S or two elements of $\overline{S}$ using comparisons between an element of S and an element of $\overline{S}$. Therefore, aside from these $n-x$ comparisons, there must be at least $c(k-1,x)$ comparisons to sort S and at least $c(k,n-x)$ comparisons to sort $\overline{S}$. This implies the following recursive inequality :

$c(k,n) \geq c(k,n-x) + n-x + c(k-1,x) >$

By the inductive assumption

$$> k(\frac{(n-x)^{1+1/k}}{e} - (n-x)) + (n-x) + (k-1)(\frac{x^{1+1/k-1}}{e} - x) =$$

By opening parentheses and permuting terms we get

$$= \frac{k}{e}(n-x)^{1+1/k} + \frac{k-1}{e} x^{1+1/k-1} + n - kn =$$

$$= \frac{k}{e} n^{1+1/k}[(1-\frac{x}{n})^{1+1/k} + (1-\frac{1}{k})\frac{x^{1+1/k-1}}{n^{1+1/k}} + \frac{1}{k}\frac{e}{n^{1/k}}] - kn \geq$$

Recall the Geometric Arithmetic Mean Inequality : $\alpha a + \beta b \geq a^{\alpha} b^{\beta}$, where $\alpha + \beta = 1 \quad \alpha, \beta, a, b \geq 0$. By taking $\alpha = 1 - \frac{1}{k}$, $\beta = \frac{1}{k}$, $a = \frac{x^{1+1/k-1}}{n^{1+1/k}}$, $b = \frac{e}{n^{1/k}}$ we get,

$$\geq \frac{k}{e} n^{1+1/k}[(1-\frac{x}{n})^{1+1/k} + \frac{x}{n^{1-1/k^2}}\frac{e^{1/k}}{n^{1/k^2}}] - kn \geq$$

Recall that the increasing sequence $(1 + \frac{1}{k})^k$ converges to *e* and therefore, $e^{1/k} > 1 + \frac{1}{k}$. This implies

$$\geq \frac{k}{e} n^{1+1/k}[(1-\frac{x}{n})^{1+1/k} + \frac{x}{n}(1+\frac{1}{k})] - kn \geq$$

Recall Bernoulli's Inequality: $(1-\alpha)^t \geq 1 - \alpha t$ for $t \geq 1 \quad \alpha \leq 1$. This implies,

$$\geq \frac{k}{e} n^{1+1/k}[1 - \frac{x}{n}(1+\frac{1}{k}) + \frac{x}{n}(1+\frac{1}{k})] - kn =$$

$$= \frac{k}{e} n^{1+1/k} - kn = k(\frac{n^{1+1/k}}{e} - n)$$

This completes the proof of the Main Theorem.

4. Upper Bound

Theorem. (due to Noga Alon) : Given n elements from a totally ordered domain, there is an algorithm in a parallel comparison model for sorting these elements in $O(\frac{logn}{log(1+p/n)})$ rounds using $p \geq n$ processors.

proof: First recall the AKS comparison network. It sorts n elements in $O(logn)$ rounds using $p = n/2$ processors (i.e., comparisons in each round). We give an algorithm in a parallel comparison model. Each round of the new algorithm is called *superround*. The algorithm is derived from AKS network by simply shrinking $\delta = 0.5 log(1+p/n)$ rounds of this network into one superround.

The construction of the algorithm is based on the following idea. We aim that the following Assertion will hold.

Assertion. After superround *r*, the following things are available: (1) The pair of input elements for each comparison performed in the first δr rounds of AKS network. (2) The result of each such comparison.

This Assertion implies that after $O(\frac{logn}{log(1+p/n)})$ superrounds the results of all comparisons of AKS network are available and the sorting is completed (since it is computable).

We show how to satisfy the Assertion for any superround *r*. For $r = 0$ the Assertion trivially holds. We show how to satisfy the Assertion for superround *r* assuming that it is satisfied for any superround $< r$.

The fact that we relate to a comparison network implies that each element, which is compared in round $\delta(r-1)+i$, where $1 \le i \le \delta$, is one of at most 2^{i-1} elements which are outputs of comparisons of the first $\delta(r-1)$ rounds (or input elements). By the inductive assumption, each of these outputs is available following superround $r-1$. Therefore, each comparison in round $\delta(r-1)+i$, is actually one of $(2^{i-1})^2$ possible pairs. All we do is perform all these possible comparisons simultaneously (for $1 \le i \le \delta$). These comparisons clearly include the actual comparisons performed by AKS network in these rounds. It remains to show that this construction also yields the pairs of input elements to each comparison which was actually performed in each of these rounds. For this we show by simple induction that the actual pair of each comparison, as well as its result are available, for all rounds $\le \delta(r-1)+i$, $0 \le i \le \delta$. For $i=0$, this follows from the inductive assumption of the Assertion for $r-1$. Suppose that for all rounds $< \delta(r-1)+i$, the actual pairs compared, as well as their result are available. Each element participating in round $\delta(r-1)+i$ is an outcome of the actual comparisons of preceding rounds and their results. They are known by the inductive assumption. Therefore, the input pair for each such comparison is known. We already argued that the result of each such comparison was found by our algorithm. This completes the proof of the induction for i. Taking $i=\delta$, we complete the inductive proof of the Assertion.

The number of comparisons that the algorithm has to perform in each superround is:

$$\frac{n}{2}\sum_{i=1}^{\delta}(2^{i-1})^2 < \frac{n}{2}\frac{4^{\delta}}{3} < \frac{n}{2}2^{2\delta}$$

But $\delta = 0.5\log(1+\frac{p}{n})$, and therefore, this number of comparisons is not more than $\frac{n}{2}2^{\log(1+\frac{p}{n})} \le \frac{n}{2}(1+\frac{p}{n}) = \frac{n+p}{2} \le p$. So there are enough processors to perform all these comparisons.

Acknowledgments. We are grateful to Noga Alon for letting us include his proof of the upper bound in this paper. Helpful comments by Noga Alon, Alan Borodin, Moshe Dubiner and Mike Paterson are gratefully acknowledged.

REFERENCES

[Ak-85] S. Akl, ''Parallel Sorting Algorithms'', Academic Press, 1985.

[AKS-83] M.Ajtai, J.Komlos and E.Szemeredi, An O(nlogn) sorting network, Proc. 15th ACM Symposium on Theory of Computation (1983) 1-9. Also, M.Ajtai, J.Komlos and E.Szemeredi, Sorting in clogn parallel steps, Combinatorica 3 (1983) 1-19.

[Al-85] N. Alon, Expanders, sorting in rounds and superconcentrators of limited depth, Proc. 17th ACM Symposium on Theory of Computing (1985) 98-102.

[BR-82] B. Bollobas and M. Rosenfeld, Sorting in one round, Israel J. Math. 38 (1981) 154-160.

[BT-83] B. Bollobas and A. Thomason, Parallel sorting, Discrete Applied Math. 6 (1983) 1-11.

[BHe-86] B. Bollobas and P. Hell, Sorting and Graphs, Preprint.

[Bo-86] B. Bollobas ,Sorting algorithms ,chapter of future monograph.

[BHo-82] A. Borodin and J.E.Hopcroft, Routing, merging and sorting on parallel models of computation, Proc. 14th ACM Symposium on Theory of Computing (1982) 338-344.

[HH-80] R. Haggkvist and P.Hell ,Graphs and parallel comparison algorithms, Congr. Num. 29 (1980) 497-509.

[HH-81] R. Haggkvist and P.Hell ,Parallel sorting with constant time for comparisons, SIAM J. Comput. 10 (1981) 465-472.

[HH-82] R. Haggkvist and P.Hell ,Sorting and merging in rounds, SIAM J. Alg. and Disc. Math. 3 (1982) 465-473.

[Kn-73] D.E. Knuth, ‘‘The Art of Computer Programming’’, Addison Wesley 1973.

[Kr-83] C.P. Kruskal , Searching, merging and sorting in parallel computation ,IEEE Trans. Computers c-32 (1983) 942-946.

[Le-84] F.T. Leighton, Tight bounds on the complexity of parallel sorting , Proc. 16th ACM Symposium on Theory of Computing (1984) 71-80.

[MW-85] F.Meyer auf der Heide and A. Wigderson, The complexity of parallel sorting, Proc. 26th IEEE Symposium on Foundations of Computer Science (1985) 532-540.

[Th-83] C. Thompson, The VLSI complexity of sorting, IEEE Trans. Computers C-32,12 (1983)

[Re-81] R. Reischuk, A fast probabilistic parallel sorting algorithm, Proc. 22nd IEEE Symposium on Foundations of Computer Science (1981) 212-219.

[RV-83] J. Reif and L.G. Valiant, A logarithmic time sort for linear size network, Proc. 15th ACM Symposium on Theory of Computation (1983) 10-16.

[SV-81] Y. Shiloach and U. Vishkin, Finding the maximum, merging and sorting in a parallel model of computation, J. Algorithms 2,1 (1981) 88-102.

[Va-75] L.G. Valiant, Parallelism in comparison problems, SIAM J. Comp. 4 (1975) 348-355.