Theoretical Computer Science 83 (1991) 275–285
Elsevier

Note

Parallel comparison merging of many-ordered lists

Y. Azar*

Department of Computer Science, Building 460, Stanford University, Stanford, CA 94305, USA

Communicated by M.S. Paterson
Received June 1989
Revised September 1990

Abstract

Azar, Y., Parallel comparison merging of many-ordered lists (Note), Theoretical Computer Science 83 (1991) 275–285.

We consider the problem of merging m disjoint ordered lists, each of size n/m. We determine up to a constant factor the worst case and average case deterministic and randomized parallel comparison complexity of the problem for all the range of n, m and p where p is the number of processors used. The worst case deterministic time complexity is

$$\Theta\left(\frac{\log m}{\log(1+p/n)}+\log\frac{\log n}{\log(2+p/n)}\right).$$

That means

$$\Theta\left(\frac{n\log m}{p}+\log\log n\right)\quad \text{for } p\leqslant 2n$$

and

$$\Theta\left(\frac{\log m}{\log(p/n)}+\log\frac{\log n}{\log(p/n)}\right)\quad \text{for } p\geqslant 2n.$$

Clearly merging two equal lists and sorting are special cases of this problem for $m=2$ and $m=n$ respectively. We also prove that these bounds hold for randomized algorithms and even for the average case of deterministic or randomized ones. Therefore the average case of the best deterministic or randomized algorithm for this problem is not faster than the worst case of the best deterministic one by more than a constant factor.

1. Introduction

Parallel comparison algorithms received a lot of attention during the last decade. The problems considered include sorting, merging, selection and their variants: [1,

* Supported by Weizmann Fellowship and by contract ONR N00014-88-K-0166.

2, 4-12, 14-16, 18-20, 22, 24, 25]. The common model of computation considered is the parallel comparison model, introduced in [25], where only comparisons are counted. In this model, during each time unit (called a *round*) a set of binary comparisons is performed. The actual set of comparisons asked is chosen according to the results of the comparisons made in previous rounds. The objective is to solve the problem at hand, trying to minimize the number of comparison rounds as well as the number of comparisons performed in each round. Note that this model ignores the time corresponding to deducing consequences from comparisons performed, as well as communication and memory addressing time. However, in some situations the comparisons cost more than the rest of the algorithm, and hence this seems to be the relevant model. Moreover, any lower bound here applies to any comparison based algorithm.

Let n denote the number of elements we have from a totally ordered domain, and suppose we have p parallel processors, i.e., we are allowed to perform p comparisons in each round. The worst case and the average case time complexity of the best deterministic or randomized algorithms for each of the basic comparison problems is known, up to a constant factor, for all admissible values of n and p. For sorting, the worst case time for deterministic algorithms is $\Theta(\log n/\log(1+p/n))$, as shown in [7, 11, 5]. As proved in [2] the same bounds hold even for the average case complexity of randomized algorithms. (See also [13] for a short proof.) For finding the maximum, the worst case deterministic time complexity is $\Theta(n/p+\log(\log n/\log(2+p/n)))$, as shown in [25], and the results of [8, 22, 10] show that the same bounds hold for general selection. For randomized algorithms or for the average case this time is just $\Theta(n/p+1)$ as was shown by [23]. Finally, the worst case complexity for merging two sorted lists, each of size n, is $\Theta(n/p+\log(\log n/\log(2+p/n)))$, as proved in [25, 12, 19]. These bounds hold also for randomized algorithms and for the average case as was shown by [16].

Merging two equal lists is a special case of the following problem. Let N be a set of n elements, partitioned into m disjoint ordered lists N_i, $|N_i| = n/m$, $i = 1, \ldots, m$. The problem of merging these lists with p processors is called *parallel merging of many ordered lists.* Clearly merging two equal ordered lists is the special case $m = 2$ and sorting is the special case where $m = n$. Moreover, this problem supplies a continuous connection between the problems of merging and sorting and, therefore, gives a better understanding of these problems, and may appear in practice in many cases. Our first result is the following:

Theorem 1.1. *Let $M(n, m, p)$ denote the worst case complexity of deterministic parallel comparison merging of m disjoint ordered lists, each of size n/m, using p processors. For all admissible n, m, p*

$$M(n, m, p) = \Theta\left(\frac{\log m}{\log(1+p/n)} + \log\frac{\log n}{\log(2+p/n)}\right).$$

Thus for any n, m and $p \leqslant 2n$,

$$M(n, m, p) = \Theta\left(\frac{n \log m}{p} + \log \log n\right)$$

and for any n, m and $p \geqslant 2n$

$$M(n, m, p) = \Theta\left(\frac{\log m}{\log(p/n)} + \log \frac{\log n}{\log(p/n)}\right).$$

Note that for $m = 2$ this becomes the bound for merging two sorted lists, each of size $n/2$. For $m = n$ the first summand in the sum of the complexity dominates and it is the sorting complexity. Throughout the paper we assume that $p < \binom{n}{2}$, as otherwise sorting can be done in one round by comparing all pairs of elements.

In order to prove Theorem 1.1, lower and upper bounds should be proved. The lower bound follows from the average case lower bound of Theorem 1.2 that we discuss later. In proving the upper bound the simple and straightforward algorithms do not give the tight upper bound and a new algorithm has to be developed. This algorithm is not based on the merging algorithm of two lists.

We say that a parallel algorithm achieves *optimal speed up* if the product of its running time by the number of processors it uses is equal, up to a constant factor, to the running time of the best serial algorithm for the same problem. I.e., if $T(n) \cdot p(n) = O(\text{Seq}(n))$, where $p(n)$ is the number of processors, $T(n)$ and $\text{Seq}(n)$ are the running times of the parallel algorithm and the best serial one, respectively, and n is the size of the input. It is easy to see that if $T'(n) > T(n)$ and there is an optimal speed up algorithm with running time $T(n)$, then there is also an optimal speed up algorithm for the same problem with running time $T'(n)$. The *parallelism break point* of a problem is the minimum $T(n)$ so that there is an optimal speed up algorithm with running time $T(n)$. A considerable amount of effort in the study of parallel algorithms is put into attempts to identify the break points of various algorithmic problems. The break point for sorting n elements (in the comparison model) is $\Theta(\log n)$, as follows from the results of [7, 5, 11]. The break point for merging two lists of size n each is $\Theta(\log \log n)$, (see [12, 19]), and the break point for selection is also $\Theta(\log \log n)$, (see [25, 8, 10]). Theorem 1.1 supplies the break points for the problem of parallel merging of many ordered lists. Specifically, it is $\Theta(\log m + \log \log n)$. Consequently, we obtain the known break points for the extreme values of m, i.e. for merging ($m = 2$) and sorting ($m = n$).

Next we consider the expected running time for randomized algorithms and the average case complexity for deterministic or randomized algorithms for this problem over all legal orders. Bounds on these cases appear to be important, since in practical situations one is naturally interested in the expected or average running time, and not necessarily in the worst case behavior. Therefore, fast randomized parallel sorting algorithms could be extremely helpful in practice. Algorithms that are fast on the average could be extremely helpful as well.

Our second result gives a negative answer for the hope of getting a better complexity for the randomized case or even for the average case. We extend the lower bound and prove the following.

Theorem 1.2. *The average comparison complexity for the best (deterministic or randomized) algorithm for parallel merging of many ordered lists is the same, up to a constant factor, as the worst case complexity of the best deterministic one.*

Note that as before, the cases where $m=2$ and $m=n$ correspond to the average complexity of merging and sorting respectively. Note also that the average parallelism break point for this problem is the same as in the worst case, up to a constant factor.

In Section 2 we prove the worst case deterministic upper bound. In Section 3 the lower bound for the average case of randomized algorithms is proved. As both bounds are of the same order of magnitude these prove Theorems 1.1 and 1.2.

2. The upper bound

In this section we prove the upper bound for $M(n, m, p)$ for all admissible n, m and p. First denote by $S(n, p)$ the worst case time for sorting n elements with p processors. The results of [7, 11, 5] determine, up to a constant factor, the complexity of $S(n, p)$ which is $\Theta(\log n/\log(1+p/n))$. This means

$$\Theta(n \log n/p) \quad \text{for } p \leq 2n \quad \text{and} \quad \Theta(\log n/\log(p/n)) \quad \text{for } p \geq 2n.$$

Our algorithm is based on the following theorem.

Theorem 2.1. *For all n, m, p in the admissible range and for $2 \leq k \leq n/m$*

$$M(n, m, p) \leq M\left(\frac{n}{k}, m, p\right) + S\left(6mk, \frac{p}{n}mk\right).$$

Proof. Divide each list N_i, $i=1, \ldots, m$ into at most $n/(mk)$ blocks of consecutive elements, each of size at most $2k$. Take the smallest element in each block to be a representing element. Form from the representing elements m ordered lists M_i, $i=1, \ldots, m$, each of size at most $n/(mk)$. Recursively merge the M_i to M, $|M| \leq n/k$, in $M(n/k, m, p)$ rounds. Now we should complete the merging of N_i to N using M.

Partition M into $n/(mk)$ sets L_j of m consecutive elements each. Define b_j^i (s_j^i) as the smallest (biggest) element of M_i which is bigger (smaller, respectively) than all the elements in L_j. Note that b_j^i, s_j^i can also be $+\infty$, $-\infty$ in the extreme cases.

Define the sets $L_j^i = \{x \in M_i \mid s_j^i \leq x \leq b_j^i\}$ and

$$A_j^i = \{x \in N_i \mid s_j^i \leq x \leq b_j^i\}, \qquad A_j = \bigcup_i A_j^i.$$

It is easy to see that $\sum_{i=1}^m |L_j^i| \leq |L_j| + 2m \leq 3m$ and $|A_j^i| \leq 2k|L_j^i|$. Therefore

$$|A_j| \leq \sum_{i=1}^m |A_j^i| \leq 2k \sum_{i=1}^m |L_j| \leq 2k \cdot 3m = 6km.$$

Assign $(pmk)/n$ processors to each A_j, $1 \leq j \leq n/(mk)$, and sort it in $S(6km, (p/n)mk)$ rounds. To complete the proof it is enough to show the following.

Proposition 2.2. *In order to sort N, using the list M, it is enough to sort each A_j, $1 \leq j \leq n/(mk)$.*

Proof. Take any two elements from different sets $x \in N_i$, $y \in N_j$ (the order relation between elements from the same set is known). Let b_x (b_y) be the smallest element of M_i (M_j) which is equal to or bigger than x (y) and let s_x (s_y) be the biggest element of M_i (M_j) which is equal to or smaller than x (y, respectively). If $b_x < s_y$ then we know that $x < y$ and if $b_y < s_x$ then we know that $y < x$ (b_x, b_y, s_x, s_y all belong to the ordered list M). Otherwise, assume without loss of generality that $b_x < b_y$, thus $s_y < b_x < b_y$. Let r be the index such that $b_x \in L_r$. Then $s_x \in L_r^i$ and hence $x \in A_r^i$. Also we claim that $b_y, s_y \in L_r^j$. This follows from the definition of L_r^j with the facts that $s_y < b_x < b_y$ and s_y and b_y are consecutive in M_j. Therefore $y \in A_r^j$, and hence $x, y \in A_r$. This completes the proof of the proposition and thus also the proof of Theorem 2.1. □

Remark. For notational simplicity, define $M(n, m, p) = 0$ for $n < m$. Thus Theorem 2.1 is true also for $k > n/m$ by performing full sorting in $S(n, p) \leq 0 + S(6mk, (p/n)mk)$.

Lemma 2.3. *For $n, m, p \geq 2n$*

$$M(n, m, p) = O\left(\frac{\log m}{\log(p/n)} + \log\frac{\log n}{\log(p/n)}\right).$$

Proof. The first step of the algorithm is Theorem 2.1 with $k = m$;

$$M(n, m, p) = M\left(\frac{n}{m}, m, p\right) + S\left(6m^2, \frac{p}{n}m^2\right) = M\left(\frac{n}{m}, m, p\right) + O\left(\frac{\log m}{\log p/n}\right).$$

Define the sequences n_i and k_i for $i \geq 0$; $n_0 = n/m$, $k_i = \lceil p/(n_i m)\rceil$, $n_i/n_{i+1} = k_i$. Note that $k_0 = \lceil p/(n_0 m)\rceil = \lceil p/n\rceil \geq 2$ and therefore it is easy to see by induction on n_i and k_i that k_i is a monotonic increasing sequence and n_i is a monotonic decreasing sequence.

The steps of the recursive algorithm are iterated applications of Theorem 2.1 with the parameters n_i, k_i;

$$\begin{aligned} M(n_i, m, p) &\leq M\left(\frac{n_i}{k_i}, m, p\right) + S\left(6m\left\lceil\frac{p}{n_i m}\right\rceil, \frac{p}{n_i}m\left\lceil\frac{p}{n_i m}\right\rceil\right) \\ &\leq M(n_{i+1}, m, p) + S\left(12\frac{p}{n_i}, \left(\frac{p}{n_i}\right)^2\right) \\ &\leq M(n_{i+1}, m, p) + O(1). \end{aligned}$$

By induction it follows that

$$M(n_0, m, p) \leq M(n_t, m, p) + O(t).$$

But

$$n_{i+1} = n_i/k_i \leqslant n_i \frac{n_i m}{p} = \frac{m}{p} n_i^2 \quad \text{or} \quad \frac{m}{p} n_{i+1} \leqslant \left(\frac{m}{p} n_i\right)^2.$$

Therefore, it follows that

$$\frac{m}{p} n_i \leqslant \left(\frac{m}{p} n_0\right)^{2^i} = \left(\frac{n}{p}\right)^{2^i}. \tag{2.1}$$

The recursive steps of the algorithm using Theorem 2.1, can proceed for t steps until $n_t < m$ (which means that everything is already sorted). Thus $t-1$ clearly satisfies $n_{t-1} \geqslant m \geqslant 1$ and by inequality (2.1) we get

$$\left(\frac{n}{p}\right)^{2^{t-1}} \geqslant \frac{m}{p} n_{t-1} \geqslant \frac{m}{p}.$$

Hence

$$2^{t-1} \log(p/n) \leqslant \log(p/m), \quad t = O\left(\log \frac{\log(p/m)}{\log(p/n)}\right) = O\left(\log \frac{\log n}{\log(p/n)}\right).$$

Thus

$$M(n, m, p) = O\left(\frac{\log m}{\log(p/n)} + \log \frac{\log n}{\log(p/n)}\right)$$

for $p \geqslant 2n$, which completes the proof. □

Lemma 2.4. *For all admissible n, m and $p \leqslant 2n$,*

$$M(n, m, p) = O\left(\frac{n \log m}{p} + \log \log n\right).$$

Proof. By Theorem 2.1, with $k = m$,

$$\begin{aligned} M(n, m, p) &\leqslant M\left(\frac{n}{m}, m, p\right) + S\left(6m^2, \frac{p}{n} m^2\right) \\ &\leqslant M\left(\frac{n}{m^2}, m, p\right) + S\left(6m^2, \frac{p}{n} m^3\right) + S\left(6m^2, \frac{p}{n} m^2\right). \end{aligned}$$

By induction it follows that for all admissible t

$$M(n, m, p) \leqslant M\left(\frac{n}{m^t}, m, p\right) + \sum_{i=1}^{t} S\left(6m^2, \frac{p}{n} m^{i+1}\right)$$

By the complexity of sorting, whenever $2 \cdot 6m^2 \geqslant (p/n)m^{i+1}$, then

$$S\left(6m^2, \frac{p}{n} m^{i+1}\right) = O\left(\frac{\log(6m^2)}{(p/n)m^{i-1}}\right)$$

and hence (assuming also $n/m^t \geq 1$)

$$M(n, m, p) \leq M\left(\frac{n}{m^t}, m, p\right) + \sum_{i=1}^{t} O\left(\frac{\log(6m^2)}{(p/n)m^{i-1}}\right)$$

$$= M\left(\frac{n}{m^t}, m, p\right) + O\left(\frac{n \log m}{p}\right)$$

Let t be defined as the biggest i which still satisfies $n \geq m^i$ and $12m^2 \geq (p/n)m^{i+1}$. If $n \geq m^i$ does not hold for $t+1$, then $n/m^t < m$ and thus $M(n/m^t, m, p) = 0$. Hence we complete the algorithm in $M(n, m, p) = O(n \log m/p)$ as needed. Otherwise $12m^2 \geq (p/n)m^{i+1}$ does not hold for $i = t+1$ and then, $(p/n)m^{t+2} \geq 12m^2$ or $n/(m^t) \leq p/12$. Hence

$$M(n, m, p) \leq M(n/m^t, m, p) + O(n \log m/p)$$

$$\leq M(p/12, m, p) + O(n \log m/p)$$

But by Lemma 2.3, $M(n', m, \Theta(n')) = O(\log m + \log \log n')$. Therefore

$$M(n, m, p) = O(\log m + \log \log p + n \log m/p) = O\left(\frac{n \log m}{p} + \log \log n\right)$$

as needed. □

The proof of the upper bound of Theorem 1.1 follows from Lemma 2.3 and Lemma 2.4.

3. The average case lower bound

In this section we prove the lower bound for the average case of deterministic and randomized algorithms. This, of course, yields also the worst case lower bound.

As observed by [26], any randomized algorithm is simply a probability distribution on (nonuniform) deterministic ones, and therefore the average complexity of the best deterministic algorithm is equal to that of the best randomized one for all n, m and p.

Denote by $AM(n, m, p)$ the average (deterministic or randomized) complexity of merging m ordered equal-size lists, each of size n/m, when the average is taken over all the $n!/(n/m)!^m$ legal orders.

Proposition 3.1. *For all* $n, m, p \leq 2n$, $AM(n, m, p) = \Omega(n \log m/p)$.

Proof. We first prove by a counting argument an $\Omega(n \log m)$ lower bound for serial algorithms. We conclude the proof of the proposition by the fact that a parallel algorithm can not speed up the serial one, by more than a factor of p (the number of processors).

It is easy to see that the number of legal orders is $n!/(n/m)!^m$. By the well-known fact that the average length of a path from a root to a leaf in a binary tree with l leaves is at least $\log_2 l$ and by the inequality $(k/e)^k \leq k! \leq ((k+1)/2)^k$ it follows that the number of steps is at least

$$\log \frac{n!}{(n/m)!^m} = \Omega\left(n \log n - m\frac{n}{m}\log\frac{n}{m}\right) = \Omega(n \log m)$$

as needed. □

Proposition 3.2

$$AM(n, m, p) = \begin{cases} \Omega(\log\log n) & \text{for } p \leq 2n \\ \Omega\left(\log \dfrac{\log n}{\log(p/n)}\right) & \text{for } p \geq 2n. \end{cases}$$

Proof. If m is even then form two disjoint sets each of size $n/2$ which consists of $m/2$ lists. Let the algorithm know for free the order in each such set. Next apply the average case lower bound for merging of [16] for the two resulting sets. It is

$$\Omega(\log\log n) \quad \text{for } p \leq 2n \quad \text{and} \quad \Omega\left(\log \frac{\log n}{\log(p/n)}\right) \quad \text{for } p \geq 2n.$$

Hence, the same bounds apply when averaging over all the legal orders of the m lists as needed.

If m is odd we form two disjoint almost equal sets (up to n/m) and use the average case lower bound of [16, journal version] for different size sets. It claims that the same lower bound holds for two sets, each of size $\Omega(n)$, as for two equal sets, each of size $\Omega(n)$. Another solution to the case where m is odd, is to let the algorithm know for free the ranks of the elements of one of the sets of size n/m and then for the remaining sets m is even again. Note that we are not allowed to assume that the known rank elements are smaller than all the other elements because we are dealing with the average case. However the lower bound proof of [16] for merging also holds when there are other elements whose ranks are known and comparisons to these elements are allowed. ([16] does not claim this but it follows from the proof). Hence we can continue as in the case where m is even and get the needed bound. □

The last part of the lower bound proof is the following.

Theorem 3.3. *For $n, m, p \geq 2n$,*

$$AM(n, m, p) = \Omega\left(\frac{\log m}{\log(p/n)}\right).$$

The lower bound proof of Theorem 1.2 and 1.1 is obtained directly by combining Propositions 3.1, 3.2 and Theorem 3.3. Thus it remains to prove Theorem 3.3.

Theorem 3.3 for $m = n$ is the main result of [2] for the average case complexity of sorting, i.e. that the average case complexity of sorting n elements requires $\Omega(\log n/\log(p/n))$ rounds using $p \geqslant 2n$ processors.

In order to prove Theorem 3.3 we use the method of Boppana who gave in [13] a short proof of the main result of [2], based on the following result of Manber and Tompa. For the sake of completeness we present the result together with its simple proof. Let an acyclic orientation of an undirected graph be directing the edges of the graph such that the resulting directed graph is acyclic, i.e., does not have directed cycles.

Theorem 3.4 (Manber et al. [21]). *The number of acyclic orientations of an undirected graph with n vertices and p edges is at most* $(1+2p/n)^n$.

Proof. Actually, we prove that this number is at most $\prod_{i=1}^{n}(1+d_i)$ where d_i is the degree of the ith vertex. For each vertex the choice of its indegree and outdegree has $1+d_i$ possibilities. Some sets of choices cannot be implemented by any acyclic orientation of our graph or even by any orientation of it. However, if a set can be implemented by an acyclic orientation then it defines uniquely one acyclic orientation, i.e. the orientation of each edge. This is done in the following way. If it defines a consistent acyclic orientation then there is a vertex with indegree zero. This defines the orientation of the edges incident with it. Omit this vertex and these edges, update the outdegree of its neighbors (subtract one; if it was previously zero then there is no consistent acyclic orientation for this possibility) and continue in the same way (or by induction). Thus the number of acyclic orientations is at most $\prod_{i=1}^{n}(1+d_i)$. By the geometric-arithmetic inequality

$$\prod_{i=1}^{n}(1+d_i) \leqslant \left[\sum_{i=1}^{n}(1+d_i)/n\right]^n = (1+2p/n)^n,$$

completing the proof. □

To prove Theorem 3.3 recall that the number of legal orders of merging m lists each of size n/m is $n!/(n/m)!^m$ and the well-known fact that the average length of a path from a root to a leaf in a tree with l leaves on which each vertex has at most t children is at least $\log l/\log t$. Therefore, in the computation tree that sorts the n elements with p processors, the number of possible answers at each step is the number of acyclic orientations of the undirected graph of the n vertices (elements) and the p edges (comparisons) which is, by Theorem 3.4, at most $(1+2p/n)^n$. Generally, it is even less, because it should be consistent with the known information

up to that step. The number of leaves in the tree is at least $n!/(n/m)!^m$ and the degree is at most $(1+2p/n)^n$. Hence the average number of steps is at least

$$\frac{\log(n!/(n/m)!^m)}{\log(1+2p/n)^n} = \Omega\left(\frac{n\log m}{n\log(1+2p/n)}\right) = \Omega\left(\frac{\log m}{\log(1+2p/n)}\right).$$

For $p \geq 2n$, this is $\Omega(\log m/(\log p/n))$ as needed. □

Acknowledgment

I would like to thank N. Alon for wonderful conversations and helpful remarks, O. Berkman for bringing the problem considered in this paper to my attention and M. Paterson for helpful remarks.

References

[1] N. Alon and Y. Azar, Sorting, approximate sorting and searching in rounds, *SIAM J. Discrete Math.* (1988) 269-280.

[2] N. Alon and Y. Azar, The average complexity of deterministic and randomized parallel comparison sorting algorithms, *SIAM J. Comput.* **6** (1988) 1178-1192. Also in: *Proc. 28th Ann. IEEE Symp. on Foundations of Computer Science*, Los Angeles (1987) 489-498.

[3] N. Alon and Y. Azar, Parallel comparison algorithms for approximation problems, in: *Proc. 29th Ann. IEEE Symp. on Foundations of Computer Science*, White Plains, New York (1988) 194-203.

[4] N. Alon and Y. Azar, Finding an approximate maximum, *SIAM J. Comput.* **2** (1989) 258-267.

[5] N. Alon, Y. Azar and U. Vishkin, Tight complexity bounds for parallel comparison sorting, in: *Proc. 27th Ann. IEEE Symp. on Foundations of Computer Science*, Toronto, Ontario, Canada (1986) 502-510.

[6] S. Akl, Parallel Sorting Algorithms (Academic Press, New York, 1985).

[7] M. Ajtai, J. Komlós and E. Szemerédi, Sorting in c log n parallel steps, *Combinatorica* **3** (1983) 1-19. Also: An O(n log n) sorting network, in: *Proc. 15th Ann. ACM Symp. on Theory of Computing* (1983) 1-9.

[8] M. Ajtai, J. Komlós, W.L. Steiger and E. Szemerédi, Deterministic selection in O(log log n) parallel time, in: *Proc. 18th Ann. ACM Symp. on Theory of Computing*, Berkeley, CA (1986) 188-195.

[9] N. Alon, Expanders, sorting in rounds and superconcentrators of limited depth, in: *Proc. 17th Ann. ACM Symp. on Theory of Computing*, Providence, Rhode Island (1985) 98-102.

[10] Y. Azar and N. Pippenger, Parallel selection, *Discrete Appl. Math.* **27** (1990) 49-58.

[11] Y. Azar and U. Vishkin, Tight comparison bounds on the complexity of parallel sorting, *SIAM J. Comput.* **3** (1987) 458-464.

[12] A. Borodin and J.E. Hopcroft, Routing, merging and sorting on parallel models of computation, *J. Comput. System Sci.* **30** (1985) 130-145. Also in: *Proc. 14th Ann. ACM Symp. on Theory of Computing*, San Francisco (1982) 338-344.

[13] R. Boppana, The average-case parallel complexity of sorting, *Inform. Process. Lett.*, **33** (1989) 145-146.

[14] B. Bollobás and A. Thomason, Parallel sorting, *Discrete Appl. Math.* **6** (1983) 1-11.

[15] R. Cole, Parallel merge sort, *SIAM J. Comput.* **17** (1988) 770-785.

[16] M. Geréb-Graus and D. Krizanc, The complexity of parallel comparison merging, in: *Proc. 28th Ann. IEEE Symp. on Foundations of Computer Science*, Los Angeles (1987) 195-201. Also *SIAM J. Comput.*, to appear.

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

[18] R. Häggkvist and P. Hell, Sorting and merging in rounds, *SIAM J. Algebraic Discrete Methods* **3** (1982) 465-473.

[19] C.P. Kruskal, Searching, merging and sorting in parallel computation, *IEEE Trans. Comput.* **32** (1983) 942-946.

[20] F.T. Leighton, Tight bounds on the complexity of parallel sorting, in: *Proc. 16th Ann. ACM Symp. on Theory of Computing*, Washington, DC (1984) 71-80.

[21] U. Manber and M. Tompa, The effect of the number of hamiltonian paths on the complexity of a vertex coloring problem, *SIAM J. Comput.* **13** (1984) 109-115.

[22] N. Pippenger, Sorting and selecting in rounds, *SIAM J. Comput.* **6** (1987) 1032-1038.

[23] R. Reischuk, A fast probabilistic sorting algorithm, in: *Proc. 22nd Ann. IEEE Symp. on Foundations of Computer Science*, Nashville, Tennessee (1981) 212-219.

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

[25] L.G. Valiant, Parallelism in comparison problems, *SIAM J. Comput.* **4** (1975) 348-355.

[26] A.C.C. Yao, Probabilistic computations: towards a unified measure of complexity, in: *Proc. 18th Ann. IEEE Symp. on Foundations of Computer Science*, Providence, Rhode Island (1977) 222-227.