

Sorting and Selection on a Linear Array with Optical Bus System

Hossam ElGindy

Dept. of Elec. & Compt. Eng.
Uni. of Newcastle, Australia

Sanguthevar Rajasekaran*

Dept. of CISE
Univ. of Florida

Abstract

In this paper we present randomized algorithms for selection and sorting on linear arrays with optical bus communication systems. We show that sorting n given numbers can be performed in $O(\log n)$ time with high probability on a linear array, of n processors, with a reconfigurable optical bus system (Linear-AROB). We also show that selection can be performed in $O(1)$ time with high probability on the same parallel machine model.

1 Introduction

Several models of parallel computing based on optical technology have been recently proposed, since the large bandwidth of optical buses has the potential to speed-up parallel computations. Examples include the Optical Communication Parallel Computer (OCPC) (see e.g., [5]), Array with Reconfigurable Optical Buses (AROB) (see e.g., [11]), and the Optical Transpose Interconnection System (OTIS) (see e.g., [8]).

*This work is supported in part by an NSF Award CCR-95-03-007 and an EPA Grant R-825-293-01-0.

Fundamental problems such as packet routing, sorting, and selection have been studied on these models. The model used in this paper is the linear version of the AROB model. A basic unit of computing in this model is called a *cycle*. A cycle consists of n *slots*, one for each processor in the array. A slot refers to the time taken by a message to traverse the length of the optical bus between two neighboring processors in the array. A cycle is the time taken by a message to traverse from processor 1 to processor n . It has been argued that for reasonably sized arrays the duration of a cycle is comparable to the time needed for a basic CPU operation and hence is assumed to be a constant [11].

Given a sequence of n numbers, the problem of sorting is to rearrange this sequence in ascending order. This problem has been extensively studied owing to its fundamental nature and importance. Several optimal algorithms have been devised in sequence (see e.g., [7]) and in parallel (see e.g., [13]).

Sorting has also been studied on the optical models [15, 6, 12, 10, 2, 3]. Sorting algorithms on 2D AROBs have been given in [15, 6]. For example, the deterministic algorithm of [15] sorts n numbers on an $n \times n^\epsilon$ 2D-AROB in $O(1)$ time, where ϵ is any constant > 0 . On the Linear-AROB, integer sorting algorithms are considered in [15, 12]. Pan et al [10] presented an algorithm with $O(\log n)$ expected time, and $O(n)$ worst case time, for general sorting on the Linear-AROB model. More recently ElGindy presented an algorithm with $O(\log^2 n)$ worst case running time on the related LAPOB model¹ [2], which was later modified to obtain an algorithm with an $O(\log n \log \log n)$ worst case time on the LAPOB [3]. In this paper we present a randomized sorting algorithm for the Linear-AROB with a run time of $O(\log n)$ with high probability. By high probability we mean a probability of $\geq (1 - n^{-\alpha})$, for any fixed $\alpha \geq 1$. We say a randomized algorithm uses $\tilde{O}(f(n))$ amount of any resource (like time, space, etc.) if the amount of resource used is no more than $cf(n)$ with probability $\geq (1 - n^{-\alpha})$ for some constant c . Similar definitions apply to $\tilde{\Theta}(\cdot)$, $\tilde{\Omega}(\cdot)$.

We also present a randomized selection algorithm that runs in $\tilde{O}(1)$ time. In contrast, the selection algorithm of Pan [9] runs in an expected $O(\log n)$ time. In the worst case this algorithm could take $\Omega(n)$ time. An $\tilde{O}(1)$ time selection algorithm

¹Linear Array with Pipelined Bus **LAPOB** is a basic optical bus model where programmable delay and reconfigurable switches are not permitted.

has been given in [15] for the 2D-AROB.

2 Preliminaries

In this section we mention some properties of the Linear-AROB that will be used subsequently. Let \mathcal{L} denote a Linear-AROB with n processors.

Problem 1. There is at most one packet originating from any node and at most one packet destined for any node. The problem is to route the packets.

Lemma 2.1 *Problem 1 can be solved in $O(1)$ cycles on \mathcal{L} .*

Proof: Each processor chooses its writing slot depending on the destination of its packet. Details can be found in [11, 15]. \square

Problem 2. There is a bit at each node. The problem is to compute the prefix sum values of these bits.

Lemma 2.2 *Problem 2 can be solved in $O(1)$ cycles in \mathcal{L} [11].*

Proof: One of the properties of the Linear-AROB is that the processors can introduce a delay of one slot in any signal passing through them. Let processor 1 initiate a signal at the beginning of a cycle. Any processor that has a one introduces a delay of one slot. Processors with zeros do not delay the signal. A processor can determine its prefix sum value by noting the time the signal reaches it. \square

Problem 3. There is a bit at each node of \mathcal{L} . let ℓ be any integer such that $1 \leq \ell \leq n$. Without loss of generality assume that ℓ divides n . The problem is to compute segmented prefix sums where each set of consecutive ℓ processors forms a segment.

Lemma 2.3 *Problem 3 can be solved in $O(1)$ cycles on \mathcal{L} .*

Proof: The proof is similar to that of Lemma 2.2. The only difference is that the first processor of each segment is now going to initiate a signal. \square

The following Lemmas are stated without proofs and are from [3].

Lemma 2.4 *A processor can broadcast a value to a contiguous set (of any size) of processors.* \square

Lemma 2.5 *Let q_1, q_2, \dots, q_k any set of k processors. Also let S_1, S_2, \dots, S_k be disjoint sets of contiguous processors that preserve the order of the k processors. Then, q_i can broadcast to S_i , for $i = 1, 2, \dots, k$, in a single cycle.* \square

Lemma 2.6 *Let q_1, q_2, \dots, q_k any set of k processors. Also let S'_1, S'_2, \dots, S'_k be disjoint sets of processors. For any S'_i ($1 \leq i \leq k$), the processors in S'_i are at equal intervals. Then, q_i can be broadcast to S'_i , for $i = 1, 2, \dots, k$, in a single cycle.* \square

3 A Logarithmic Time Sorting Algorithm

In this section we present an $\tilde{O}(\log n)$ -cycle algorithm for sorting n elements on \mathcal{L} .

Before presenting the algorithm, we show that sorting can be done in $O(1)$ cycles in the worst case on \mathcal{L} provided there are only \sqrt{n} elements. Let the elements to be sorted be $k_1, k_2, \dots, k_{\sqrt{n}}$ and let them be in the processors $1, 2, \dots, \sqrt{n}$. This algorithm is similar to the merging algorithm given in [3].

Let S_1 be the set of the first \sqrt{n} processors of \mathcal{L} , S_2 be the next \sqrt{n} processors of \mathcal{L} , and so on. Also let S'_1 be the set of processors consisting of $1, \sqrt{n} + 1, 2\sqrt{n} + 1, \dots$, S'_2 be the set consisting of $2, \sqrt{n} + 2, 2\sqrt{n} + 2, \dots$, etc. There are four steps in the algorithm.

Step 1. Broadcast k_1 to S'_1 , k_2 to S'_2 , and so on. These broadcasts can be done in $O(1)$ cycles (c.f. Lemma 2.6). At the end of this, each set S_i (for $1 \leq i \leq \sqrt{n}$) has a copy of the input sequence.

Step 2. Broadcast k_1 to S_1 , k_2 to S_2 , and so on. This takes $O(1)$ cycles (c.f. Lemma 2.5). Processors in S_1 compare k_1 with every input key; processors in S_2 compare k_2 with every input key; etc.

Step 3. Perform a segmented prefix sums computation (c.f. Problem 3) with $\ell = \sqrt{n}$ so that the rank of each input key can be computed.

Step 4. Route the key whose rank is i to processor i , for $1 \leq i \leq \sqrt{n}$. This takes $O(1)$ cycles in accordance with Lemma 2.1.

The correctness of this algorithm is quite clear. Thus we get the following Theorem.

Theorem 3.1 *We can sort \sqrt{n} keys in $O(1)$ cycles on \mathcal{L} . □*

Now we are ready to describe our randomized algorithm. The technique of sampling has dominated the design of both sequential and parallel sorting algorithms in the past two decades. A popular technique is one that was introduced by Frazer and McKellar [4]. The idea is to pick a random sample S of s keys from the input, sort this input, partition the input using the sorted sample keys as splitter keys, and sort the resultant parts independently. This approach has been used over a variety of parallel models. The following sampling lemma from [14] will be useful in the analysis of our algorithm. Let $S = \{k_1, k_2, \dots, k_s\}$ be a random sample from a set Y of cardinality N . Let ‘select(Y, i)’ stand for the i th smallest element of Y for any integer i . Also let k'_1, k'_2, \dots, k'_s be the sorted order of the sample S . If r_i is the rank of k'_i in Y and if $|S| = s$, the following lemma [14] provides a high probability confidence interval for r_i .

Lemma 3.1 *For every $\alpha > 0$, Prob. $(|r_i - i\frac{N}{s}| > \sqrt{3\alpha}\frac{N}{\sqrt{s}}\sqrt{\log N}) < N^{-\alpha}$.*

Our algorithm differs from that of Frazer and McKellar’s. We still use random sampling. Whereas Frazer and McKellar’s algorithm partitions the input using s splitter keys (where s is typically chosen to be $\Omega(\sqrt{n})$), we partition the input using a single splitter key. Also, Frazer and McKellar’s algorithm randomly chooses the s splitter keys. In contrast we choose a random sample and use the median of this sample as the splitter key.

In particular, we pick a random sample S of size $\tilde{\Theta}(n^{1/4})$, sort this sample using Theorem 3.1 and as a result find the median M of this sample, partition the input around this median, and recursively sort the resultant parts in parallel. At any given

time the array will consist of subproblems where a subproblem will be that of sorting keys in a contiguous set of processors. To begin with there will be only one subproblem (of size n).

Consider any subproblem that spans processors i through j . Let $N = (j - i + 1)$. The following algorithm is executed for each such subproblem in parallel.

Algorithm Sort

repeat

Step 1. Each processor decides to include its key in the sample S with probability $\frac{1}{N^{3/4}}$. The expected number of sample keys is $N^{1/4}$. Using Chernoff bounds [1] we can show that the number of keys in S is $\tilde{\Theta}(N^{1/4})$. (Here the underlying probability is $\geq (1 - N^{-\alpha})$.)

Step 2. Using a prefix computation count the number s of keys in S . Also using Lemma 2.1 compact the sample keys in the first s processors.

Step 3. Sort the sample keys using Theorem 3.1. Thus find and broadcast the median M of the sample keys.

Step 4. Let N_1 be the number of keys in the subproblem that are less than or equal to M and let N_2 be the number of keys greater than M . Rearrange the keys so that all the keys that are less than or equal to M will appear first followed by all the keys that are greater than M . The subproblem has thus been replaced with two new subproblems, one from i to $i + N_1 - 1$ and the other from $i + N_1$ to j .

until the maximum size of any subproblem is $O(1)$; Sort the subproblems.

Theorem 3.2 *Algorithm Sort runs in $\tilde{O}(\log n)$ cycles.*

Proof: The main idea behind the proof is to show that the size of any subproblem reduces by a factor of nearly two in one iteration of the *repeat* loop (with high probability) and that each iteration of the *repeat* loop only takes $O(1)$ cycles.

Step 1 takes $O(1)$ time. Step 2 involves a prefix computation and a partial permutation routing. Thus Step 2 takes $O(1)$ cycles (c.f. Lemmas 2.2 and 2.1).

Step 3 involves sparse enumeration sorting and hence can be completed in $O(1)$ cycles (in accordance with Theorem 3.1). In Step 4, we can count N_1 using Lemma 2.2 in $O(1)$ cycles. As a byproduct we can also compute a unique address for each key in order to perform the compaction step. Routing can be done using Lemma 2.1. Therefore, Step 4 also takes $O(1)$ cycles.

We can check if the maximum size of any subproblem is $O(1)$ or not using a prefix computation as follows. Let a subproblem span the processors i through j . Form a sequence in which the $(i + 1)$ th through the j th elements are zeros. The i th element is one if the size of this subproblem is greater than some constant and zero otherwise. Perform a prefix computation in the whole array to count the number of subproblems that are greater than a constant. Once the *repeat* loop terminates it is easy to sort the sub-problems in $O(1)$ time.

In summary, each iteration of the *repeat* loop can be completed in $O(1)$ cycles. It remains to be shown that there will only be $\tilde{O}(\log n)$ iterations of the *repeat* loop.

Consider any subproblem X of size N . Each element is chosen to be in the sample S with probability $\frac{1}{N^{3/4}}$. Thus the expected number of sample keys is $N^{1/4}$. Using Chernoff bounds, this number is seen to be $s = \tilde{\Theta}(N^{1/4})$.

The median M of S is expected to be an approximate median of the subproblem. Using the sampling Lemma 3.1 we see that the rank of M in X can differ from $\frac{N}{2}$ by at most $\sqrt{3.5\alpha} \frac{N}{\sqrt{s}} \sqrt{\log N} = O(N^{0.9})$. In other words, M is a very close approximation to the median of X .

Thus a given subproblem gets split into two subproblems in an iteration of the *repeat* loop such that each such new subproblem is of size no more than $0.6N$ with probability $\geq (1 - N^{-\alpha})$. As a consequence it follows that the number of iterations of the *repeat* loop is $\tilde{O}(\log n)$. Note that this expectation is over the space of all possible outcomes for the coin flips made in the algorithm (and not over the space of all possible inputs).

Since the probability of our assertion decreases with iteration, it's not clear if the number of iterations will be $O(\log n)$ with probability $\geq (1 - n^{-\alpha})$. There are a number of ways in which we can indeed show that the number of iterations is only $O(\log n)$ with probability $\geq (1 - n^{-\alpha})$. One way is to employ the process tree defined in [16]. We omit the details. \square

4 A Constant Time Selection Algorithm

Selection is an important comparison problem that has numerous applications. Selection takes as input a sequence of n keys and an integer $i, 1 \leq i \leq n$. The goal is to identify the i th smallest of the n input keys. Several linear time algorithms are known for sequential selection (see e.g., [7]), and optimal algorithms have also been devised on various parallel models (see e.g., [13]). In this section we show that selection from n keys can be performed in $\tilde{O}(1)$ cycles on an n -node Linear-AROB. The idea is to make use of sampling as follows:

1. Choose a random sample S of size $q = o(n)$
2. Identify two elements ℓ_1 and ℓ_2 from the sample whose ranks in S are $i\frac{q}{n} - \delta$ and $i\frac{q}{n} + \delta$ for some appropriate δ . It can be shown that these elements ‘bracket’ the element to be selected with high probability;
3. Eliminate all keys whose values are outside the range $[\ell_1, \ell_2]$;
4. Perform an appropriate selection from out of the remaining keys.

In the case of Linear-AROB, the above idea can be implemented as follows. The algorithm starts with the number of *alive* keys N being the same as n and progresses in *stages*, where each stage proceeds as follows:

1. If $N \leq \sqrt{n}$, compact the alive keys, sort them using Theorem 3.1, and output the i th smallest key.
2. Each processor decides to include its key in the sample S with probability $\frac{1}{N^{3/4}}$. The number of keys in S is $\tilde{\Theta}(N^{1/4})$.

3. Compact and sort the sample keys using Theorem 3.1. Let ℓ_1 and ℓ_2 be the keys from S with ranks $\lceil \frac{i|S|}{N} \rceil - d\sqrt{|S|\log N}$ and $\lceil \frac{i|S|}{N} \rceil + d\sqrt{|S|\log N}$, respectively, d being a constant $> \sqrt{3\alpha}$. We can show that ℓ_1 and ℓ_2 will bracket the i th smallest element with high probability.
4. Now kill the alive keys whose values are not in the interval $[\ell_1, \ell_2]$. Let N be the number of surviving keys. Let N_1 be the number of keys that got killed with a value $< \ell_1$.
5. If the key to be selected does not have a value in the interval $[\ell_1, \ell_2]$ redo this stage else set $i = i - N_1$ and proceed to the next stage.

It is easy to see that each step in a stage can be performed in $O(1)$ cycles and hence a stage itself takes only $O(1)$ cycles. Also we can show using Lemma 3.1 that if N is the number of alive keys at the beginning of any stage, then the number of alive keys at the end of the stage is $\tilde{O}(N^{0.9})$. This in turn means that there can only be $\tilde{O}(1)$ stages in the algorithm. Thus we get the following Theorem.

Theorem 4.1 *We can perform selection from out of n keys in $\tilde{O}(1)$ cycles on an n -node Linear-AROB.*

5 Conclusions

In this paper we have presented an $\tilde{O}(\log n)$ cycles algorithm to sort n given numbers on an n -node Linear-AROB. It is still an open problem to match these bounds using a deterministic algorithm. We have also presented an $\tilde{O}(1)$ cycles selection algorithm for the Linear-AROB model.

References

- [1] H. Chernoff, A Measure of Asymptotic Efficiency for Tests of a Hypothesis Based on the Sum of Observations, *Annals of Mathematical Statistics* 23, 1952, pp. 493-507.
- [2] H. ElGindy, Sorting on Linear Arrays with Optical Buses, Manuscript, February 1998.
- [3] H. ElGindy, An Improved Sorting Algorithm for Linear Arrays with Optical Buses, submitted for publication, April 1998.
- [4] W.D. Frazer and A.C. McKellar, Samplesort: A Sampling Approach to Minimal Storage Tree Sorting, *Journal of the ACM* 17(3), 1970, pp. 496-507.
- [5] L. Goldberg, M. Jerrum, T. Leighton, and S. Rao, A Doubly-Logarithmic Communication Algorithm for the Completely Connected Optical Communication Parallel Computer, Proc. Symposium on Parallel Algorithms and Architectures, 1993, pp.300-309.
- [6] Z. Guo, Sorting on Array Processors with Pipelined Buses, *Proc. International Conference on Parallel Processing*, Volume III, 1992, pp. 289-292.
- [7] E. Horowitz, S. Sahni, and S. Rajasekaran, *Computer Algorithms*, W.H. Freeman Press, 1998.
- [8] G. C. Marsden, P. J. Marchand, P. Harvey, and S. C. Esener, Optical Transpose Interconnection System Architectures, *Optic Letters*, 18(3):1083-1085, July 1993.
- [9] Y. Pan, "Order statistics on optically interconnected multiprocessor system," The First International Workshop on Massively Parallel Processing Using Optical Interconnections, Cancun, Mexico, pp. 162 – 169, April 1994.
- [10] Y. Pan and M. Hamdi, "Quicksort on a linear array with a reconfigurable pipelined bus system," The IEEE International Symposium on Parallel Architectures, Algorithms, and Networks, Beijing, China, June 1996.

- [11] S. Pavel and S. Akl, On the Power of Arrays with Optical Pipelined Buses, *Proc. International Conference on Parallel and Distributed Processing Techniques and Applications*, Vol. III, 1996, pp. 1443-1454.
- [12] S. Pavel and S. Akl, Integer Sorting and Routing in Arrays with Reconfigurable Optical Buses, *Proc. International Conference on Parallel Processing*, 1996.
- [13] S. Rajasekaran, Sorting and Selection on Interconnection Networks, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* 21, 1995, pp. 275-296.
- [14] S. Rajasekaran and J.H. Reif, Derivation of Randomized Algorithms for Sorting and Selection, in *Parallel Algorithm Derivation And Program Transformation*, edited by R. Paige, J.H. Reif, and R. Wachter, Kluwer Academic Publishers, 1993.
- [15] S. Rajasekaran and S. Sahni, Sorting, Selection, and Routing on the Array with Reconfigurable Optical Buses, *IEEE Transactions on Parallel and Distributed Systems* 8(1), 1997, pp. 1123-1132.
- [16] S. Rajasekaran and S. Sen, Random Sampling Techniques and Parallel Algorithms Design, in *Synthesis of Parallel Algorithms*, edited by J.H. Reif, Morgan-Kaufmann Publishers, 1993.