

SORTING AND SELECTION ON INTERCONNECTION NETWORKS

SANGUTHEVAR RAJASEKARAN

ABSTRACT. In this paper we identify techniques that have been employed in the design of sorting and selection algorithms for various interconnection networks. We consider both randomized and deterministic techniques. Interconnection Networks of interest include the mesh, the mesh with fixed and reconfigurable buses, the hypercube family, and the star graph. For the sake of comparisons, we also list PRAM algorithms.

1 Introduction

The problem of sorting a given sequence of n keys is to rearrange this sequence in nondecreasing order. Given a sequence X of n keys, and an integer i ($1 \leq i \leq n$), the problem of selection is to find the i th smallest key in the sequence. Such a key will be denoted as $\text{select}(i, X)$.

These two important comparison problems have been studied extensively by computer scientists. Both sorting and selection have asymptotically optimal sequential algorithms. There are several sorting algorithms that run in time $O(n \log n)$ in the worst case (see e.g., [1]). The problem of selection has an elegant linear time sequential algorithm [8].

1991 Mathematics Subject Classification: Primary: 68P10, 68Q22; Secondary: 68Q25, 62D05.

Keywords and phrases: Sorting, Selection, Parallel Algorithms.

This research was supported in part by an NSF Research Initiation Award CCR-92-09260.

Optimal algorithms for these two problems also exist for certain parallel models like the EREW PRAM, the comparison tree model, etc. This paper deals with the solution of sorting and selection on interconnection networks. We show how the idea of sampling (both randomized and deterministic) has been repeatedly used to obtain optimal or near optimal algorithms for these problems.

1.1 Models Definition

The Mesh. A mesh connected computer (referred to as the Mesh from hereon) is a $\sqrt{p} \times \sqrt{p}$ square grid where there is a processor at each grid point. Each processor is connected to its four or less neighbors through bidirectional links. It is assumed that in one unit of time a processor can perform a local computation and/or communicate with all its neighbors.

A Mesh with Buses. Two variants of the Mesh have attracted the attention of many a researcher lately: 1) the mesh connected computer with fixed buses (denoted as M_f), and 2) the Mesh with reconfigurable buses (denoted as M_r).

In M_f we assume that each row and each column has been augmented with a broadcast bus. Only one message can be broadcast along any bus at any time, and this message can be read by all the processors connected to this bus in the same time unit.

In the model M_r , processors are connected to a reconfigurable broadcast bus. At any given time, the broadcast bus can be partitioned (i.e., reconfigured) dynamically into subbuses with the help of locally controllable switches. Each processor has 4 I/O ports. There are many variations of M_r found in the literature. In PARBUS model, any combination of 4 port connections is permitted for each processor [22]. Each subbus connects a collection of successive processors. One of the processors in this collection can choose to broadcast a message which is assumed to be readable in one unit of time by all the other processors in this collection.

For instance, in a $\sqrt{p} \times \sqrt{p}$ mesh, the different columns (or different rows) can form subbuses. Even within a column (or row) there could be many subbuses, and so on. It is up to the algorithm designer to decide what configuration of the bus should be used at each time unit. The model assumed in this paper is essentially the same as PARBUS.

Both M_r and M_f are becoming popular models of computing because of the absence of diameter consideration and because of the commercial

implementations [5, 40, 62, 35]. Even as theoretical models, both M_r and M_f are very interesting. For instance, n keys can be sorted in $O(1)$ time on an $n \times n$ mesh M_r , whereas we know that $\Omega(\frac{\log n}{\log \log n})$ time is needed even on the CRCW PRAM given only a polynomial number of processors.

The Hypercube. A hypercube of dimension ℓ consists of $p = 2^\ell$ nodes (or vertices) and $\ell 2^{\ell-1}$ edges. Thus each node in the hypercube can be named with an ℓ -bit binary number. If x is any node in V , then there is a bidirectional link from x to a node y if and only if x and y (considered as binary numbers) differ in exactly one bit position (i.e., the hamming distance between x and y is 1.) Therefore, there are exactly ℓ edges going out of (and coming into) any vertex.

If a hypercube processor can handle only one edge at any time step, this version of the hypercube will be called the *sequential model*. Handling (or processing) an edge here means either sending or receiving a key along that edge. A hypercube model where each processor can process all its incoming and outgoing edges in a unit step is called the *parallel model*.

Star Graphs Let $s_1 s_2 \dots s_n$ be a permutation of n symbols, e.g., $1 \dots n$. For $1 < j \leq n$, we define $SWAP_j(s_1 s_2 \dots s_n) = s_j s_2 \dots s_{j-1} s_1 s_{j+1} \dots s_n$. An n -star graph is a graph $S_n = (V, E)$ with $|V| = n!$ nodes, where $V = \{s_1 s_2 \dots s_n \mid s_1 s_2 \dots s_n \text{ is a permutation of } n \text{ different symbols}\}$, and $E = \{(u, v) \mid u, v \in V \text{ and } v = SWAP_j(u) \text{ for some } j, 1 < j \leq n\}$.

A Notation. Throughout this paper we let n denote the input size and p denote the number of processors available.

2 Randomized and Deterministic Sampling

Sampling has served as an effective tool in the design of sorting and selection algorithms over a variety of models of computing. The use of sampling in comparison problems is at least as old as that of Frazer and McKellar's paper [16]. The following strategy has been proposed for sequential sorting in [16]: 1) Randomly sample $o(n)$ keys from the input and sort them using any nonoptimal algorithm; 2) Partition the input into independent subsequences using the sample keys as splitter elements; and 3) Sort each subsequence separately. Clearly, this approach can be thought of as a generalization of Hoare's quicksort algorithm [20] where a single key is employed to partition

the input. This elegant approach of Frazer and McKellar has been adapted to design sorting algorithms on various interconnection networks as will be shown in this paper.

Random sampling has also played an important role in the development of efficient algorithms for selection. For example, Floyd and Rivest [15] have introduced the following scheme for sequential selection: 1) Sample $o(n)$ keys from the input and pick two elements from the sample (call them ℓ_1 and ℓ_2) such that the element to be selected has a value in the range $[\ell_1, \ell_2]$ and the number of input keys in the range $[\ell_1, \ell_2]$ is ‘small’; 2) Eliminate all the input keys that fall outside the range $[\ell_1, \ell_2]$; and 3) Perform an appropriate selection from out of the remaining keys (the remaining keys can even be sorted to perform this selection). The sequential run time of this algorithm can be shown to be $n + \min\{i, n - i\} + o(n)$ with high probability, if we want to select the i th smallest key.

Recently various selection and sorting algorithms (both deterministic and randomized) have been implemented on different parallel machines. These experimental results indicate that randomized algorithms perform better in practice than their deterministic counterparts (see e.g., [7], [19], [61], [51]).

The idea of sampling has been employed in deterministic algorithms as well. Classical examples are the selection algorithms of 1) Blum, Floyd, Pratt, Rivest, and Tarjan [8]; and 2) Munro and Paterson [41]. A synopsis of deterministic sampling is: 1) To group the numbers into groups of ℓ numbers each (for an appropriate ℓ); 2) Sort each group independently; 3) Collect every q th element from each group (for some q). This collection serves as a ‘sample’ for the original input. For example, the median of this sample can be shown to be an approximate median for the input. Deterministic sampling has also found numerous applications as will be demonstrated in this paper.

2.1 Sampling Lemmas

Random Sampling. Let X be a sequence of n numbers from a linear order and let $S = \{k_1, k_2, \dots, k_s\}$ be a random sample from X . Also let k'_1, k'_2, \dots, k'_s be the sorted order of this sample. If r_i is the rank of k'_i in X , the following lemma provides a high probability confidence interval for r_i . (The rank of any element k in X is the number of elements $\leq k$ in X .)

Lemma 2.1 *For every α , Prob. $\left(|r_i - i\frac{n}{s}| > \sqrt{3\alpha} \frac{n}{\sqrt{s}} \sqrt{\log n}\right) < n^{-\alpha}$.*

A proof of the above Lemma can be found in [54]. This lemma can be used to analyze many of the selection and sorting algorithms based on random sampling.

A Notation. We say a randomized algorithm has a resource bound of $\tilde{O}(f(n))$ if there exists a constant c such that the amount of resource used is no more than $caf(n)$ **on any input** of size n with probability $\geq (1 - n^{-\alpha})$ (for any $\alpha > 0$). In an analogous manner, we could also define the functions $\tilde{o}(\cdot)$, $\tilde{\Omega}(\cdot)$, etc.

Deterministic Sampling. Consider a sequence X of n numbers. A deterministic sample S from X is obtained as follows: Partition the numbers into groups with ℓ elements in each group. Pick every q th element from each group. There will be roughly $\frac{\ell}{q}$ elements picked from each group and $\frac{n}{q}$ elements in all. Let k'_i be the element of S whose rank (in S) is i . Let r_i be the rank of k'_i in X . Then, the following Lemma is easy to prove:

Lemma 2.2 r_i is in the range $[iq, iq + \frac{n}{\ell}q]$.

2.2 Organization of this Paper

The rest of this paper is organized as follows. In Section 3, we survey known sorting algorithms on the network models of our interest. In Section 4, we provide a summary of known selection algorithms and in Section 5 we provide a list of open problems.

3 Sorting on Interconnection Networks

3.1 A Generic Algorithm

Algorithm I

Step 1. Randomly sample n^ϵ (for some constant $\epsilon < 1$) keys.

Step 2. Sort this sample (using any nonoptimal algorithm).

Step 3. Partition the input using the sorted sample as splitter keys.

Step 4. Sort each part separately in parallel.

3.2 The PRAM

One of the classical results in parallel sorting is Batcher's algorithm [3]. This algorithm is based on the idea of bitonic sorting and was proposed for the hypercube and hence can be run on any of the PRAMs as well. Batcher's algorithm runs in $O(\log^2 n)$ time using n processors. Followed by this, a very nearly optimal algorithm was given by Preparata [47]. Preparata's algorithm used $n \log n$ processors and took $O(\log n)$ time. Finding a logarithmic time optimal parallel algorithm for sorting remained an open problem for a long time in spite of numerous attempts. Finally in 1981, Reischuk was able to design a randomized logarithmic time optimal algorithm for the CREW PRAM [58]. At around the same time Ajtai, Komlós, and Szemerédi announced their sorting network of depth $O(\log n)$ [2]. However the size of the circuit was $O(n \log n)$ and also the underlying constant in the time bound was enormous. Leighton subsequently was able to reduce the circuit size to $O(n)$ using the technique of columnsort [33]. Though several attempts have been made to improve the constant in the time bound, the algorithm of [2] remains vastly as a result of theoretical interest.

In 1987 Cole presented an optimal logarithmic time EREW PRAM algorithm for sorting, the constant in the time bound being reasonably small [10]. In the same paper, a sub-logarithmic time algorithm for sorting on the CRCW PRAM is also given. This algorithm uses $n(\log n)^\epsilon$ processors, the run time being $O(\frac{\log n}{\log \log \log n})$. Here ϵ is any constant > 0 . The lower bound result of Beam and Hastad states that any CRCW PRAM sorting algorithm will have to take $\Omega(\frac{\log n}{\log \log n})$ time in the worst case as long as the processor bound is only a polynomial in the input size n [4]. Rajasekaran and Reif [53] were able to obtain a randomized algorithm for sorting on the CRCW PRAM that runs in time $O(\frac{\log n}{\log \log n})$, the processor bound being $n(\log n)^\epsilon$, for any fixed $\epsilon > 0$. This algorithm is also processor-optimal, i.e., to achieve the same time bound the processor bound can not be decreased any further.

Table 1 summarizes these algorithms.

3.3 The Mesh

The first asymptotically optimal sorting algorithm for the mesh was given by Thompson and Kung [63]. Their algorithm can sort n numbers on a $\sqrt{n} \times \sqrt{n}$ Mesh in $O(\sqrt{n})$ time. Since the diameter of an n -node Mesh is $2\sqrt{n} - 2$, [63]'s algorithm is clearly optimal. Thompson and Kung's algorithm is based on the idea of odd-even merging. Since a Mesh has a large diameter, it is

AUTHOR(S)	YEAR	MODEL	P	T
Preparata [47]	1978	CREW	$n \log n$	$O(\log n)$
Reischuk [58]	1981	CREW	n	$\tilde{O}(\log n)$
Cole [10]	1987	EREW	n	$O(\log n)$
Cole [10]	1987	CRCW	$n(\log n)^\epsilon$	$O(\frac{\log n}{\log \log \log n})$
Rajasekaran & Reif [53]	1988	CRCW	$n(\log n)^\epsilon$	$\tilde{O}(\frac{\log n}{\log \log n})$

Table 1: Sorting on the PRAM

imperative to have not only asymptotically optimal algorithms but also they should have small underlying constants in their time bounds. Often times, the challenge in designing Mesh algorithms lies in reducing the constants in time bounds.

Subsequent to Thompson and Kung's algorithm, Schnorr and Shamir gave a $3\sqrt{n} + o(\sqrt{n})$ time algorithm [59]. They also proved a lower bound of $3\sqrt{n} - o(\sqrt{n})$ for sorting. However, both the upper bound and the lower bound were derived under the assumption of no queueing. Ma, Sen, and Scherson [37] gave a near optimal algorithm for a related model. Kaklamanis, Krizanc, Narayanan, and Tsantilas presented a very interesting algorithm for sorting with a run time of $2.5\sqrt{n} + o(\sqrt{n})$ [25]. This algorithm was randomized and used queues of size $O(1)$. The underlying idea here is the same as that of Algorithm I. The same authors latter improved this time bound to $2\sqrt{n} + \tilde{o}(\sqrt{n})$ [24].

The idea of using $O(1)$ sized queues has been successfully employed to design better deterministic sorting algorithms as well. Kunde has presented a $2.5\sqrt{n} + o(\sqrt{n})$ step algorithm [31]; Nissam and Sahni have given a $(2 + \epsilon)\sqrt{n} + o(\sqrt{n})$ time algorithm (for any fixed $\epsilon > 0$) [44]; Also Kaufmann, Sibeyn, and Torsten have offered a $2\sqrt{n} + o(\sqrt{n})$ time algorithm [26]. The third algorithm closely resembles the one given by [25] and Algorithm I.

The problem of $k - k$ sorting is to sort a Mesh where k elements are input at each node. The bisection lower bound for this problem is $\frac{k\sqrt{n}}{2}$. For example, if we have to interchange data from one half of the mesh with data from the other half, $\frac{k\sqrt{n}}{2}$ routing steps will be needed. A very nearly optimal randomized algorithm for $k - k$ sorting is given in [49]. Recently, Kunde [32] has matched this result with a deterministic algorithm.

Table 2 summarizes sorting results for the Mesh.

AUTHOR(S)	YEAR	PROBLEM	TIME
Thompson & Kung [63]	1977	1 – 1 Sort	$7\sqrt{n} + o(\sqrt{n})$
Schnorr & Shamir [59]	1986	1 – 1 Sort	$3\sqrt{n} + o(\sqrt{n})$
Ma, Sen, & Scherson [37]	1986	1 – 1 Sort	$4\sqrt{n} + o(\sqrt{n})$
KKNT [25]	1991	1 – 1 Sort	$2.5\sqrt{n} + \tilde{o}(\sqrt{n})$
Kunde [31]	1991	1 – 1 Sort	$2.5\sqrt{n} + o(\sqrt{n})$
KKNT [25]	1992	1 – 1 Sort	$2\sqrt{n} + \tilde{o}(\sqrt{n})$
Nigam & Sahni [44]	1993	1 – 1 Sort	$(2 + \epsilon)\sqrt{n} + o(\sqrt{n})$
KST [26]	1993	1 – 1 Sort	$2\sqrt{n} + o(\sqrt{n})$
Kunde [31]	1991	$k - k$ Sort	$k\sqrt{n} + o(k\sqrt{n})$
Rajasekaran [49]	1991	$k - k$ Sort	$\frac{k\sqrt{n}}{2} + 2\sqrt{n} + \tilde{o}(k\sqrt{n})$

Table 2: Mesh Algorithms for Sorting

3.4 Meshes with Buses

For a Mesh with fixed buses, it is easy to design a logarithmic time algorithm for sorting n numbers using a polynomial (in n) number of processors (see e.g., [50]). However, if the Mesh is of size $\sqrt{n} \times \sqrt{n}$, then the bisection lower bound for sorting will be $\Omega(\sqrt{n})$. The same lower bound holds for a Mesh with a reconfigurable bus system also. In general, we can obtain impressive speedups on M_r and M_f if the number of processors used is much more than the input size.

When the input size n is the same as that of the network size, sorting can be done using a randomized algorithm on M_r in time that is only $\tilde{o}(\sqrt{n})$ more than the time needed for packet routing under the same settings as has been proven in [52]. This randomized algorithm is also similar to Algorithm I. In [29], Krizanc, Rajasekaran, and Shende show that on M_f also, sorting can be done in time that is nearly the same as the time needed for packet routing. The best known algorithm for packet routing on M_r takes time $\frac{17}{18}\sqrt{n} + \tilde{o}(\sqrt{n})$ [9]. For M_f , the best known packet routing time is $0.79\sqrt{n} + \tilde{o}(\sqrt{n})$ [60]. Therefore, sorting can be done on M_r in time $\frac{17}{18}\sqrt{n} + \tilde{o}(\sqrt{n})$ and on M_f in time $0.79\sqrt{n} + \tilde{o}(\sqrt{n})$.

An interesting feature of M_r is that sorting can be done on it in time $O(1)$ using a quadratic number of processors. In contrast, sorting can not be done in $O(1)$ time even on the CRCW PRAM, given only a polynomial

AUTHOR(S)	YEAR	MODEL	P	T
Batcher [3]	1968	Network	n	$\frac{1}{2} \log^2 n$
Nassimi & Sahni [43]	1981	Hypercube	$n^{1+\epsilon}$	$O(\log n)$
Reif & Valiant [57]	1983	CCC	n	$\tilde{O}(\log n)$
Cypher & Plaxton [12]	1990	Hypercube	n	$O(\log n \log \log n)$

Table 3: Sorting on the Hypercube

number of processors [4]. A constant time algorithm using n^3 processors appears in [64]. The processor bound was improved to n^2 in independent works [23], [36], [42], [45].

3.5 The Hypercube

Batcher's algorithm runs in $O(\log^2 n)$ time on an n -node hypercube [3]. This algorithm uses the technique of bitonic sorting. Odd-even merge sorting can also be employed on the hypercube to obtain the same time bound. Nassimi and Sahni [43] gave an elegant $O(\log n)$ time algorithm for sorting which uses $n^{1+\epsilon}$ processors (for any fixed $\epsilon > 0$). This algorithm, known as *sparse enumeration sort*, has found numerous applications in the design of other sorting algorithms on various interconnection networks. A variant of Algorithm I was employed by Reif and Valiant to derive an optimal randomized algorithm for sorting on the CCC [57]. The best known deterministic algorithm for sorting on the hypercube (or any variant) is due to Cypher and Plaxton and it takes $O(\log n \log \log n)$ time [12]. This algorithm makes use of the technique of deterministic sampling and the underlying constant in the time bound is rather large. An excellent description of this algorithm can be found in [34]. See Table 3.

3.6 The Star Graph

Menn and Somani [39] employed an algorithm similar to that of Schnorr and Shamir [59] to show that sorting can be done on a star graph with $n!$ nodes in $O(n^3 \log n)$ time. Rajasekaran and Wei [56] have offered a randomized algorithm with a time bound of $\tilde{O}(n^3)$. This algorithm is based on a randomized selection algorithm that they derive. A summary of this algorithm follows:

AUTHOR(S)	YEAR	P	T
Menn and Somani [39]	1990	$n!$	$O(n^3 \log n)$
Rajasekaran and Wei [56]	1992	$n!$	$\tilde{O}(n^3)$

Table 4: Sorting on the Star Graph

There are n phases in the algorithm. A star graph with $n!$ nodes is denoted as S_n . In the first phase they perform a selection of n uniformly distributed keys and as a consequence route each key to the correct sub-star graph S_{n-1} it belongs to. In the second phase, sorting is local to each S_{n-1} . At the end of second phase each key will be in its correct S_{n-2} . In general, at the end of the ℓ th phase, each key will be in its right $S_{n-\ell}$ (for $1 \leq \ell \leq n-1$). Selection in each phase takes $\tilde{O}(n^2)$ time. Making use of these selected keys, every input key figures out the $S_{n-\ell}$ it belongs to in $O(n^2)$ time. The keys are routed to the correct $S_{n-\ell}$'s in $\tilde{O}(n)$ time. Thus each phase takes $\tilde{O}(n^2)$ time, accounting for a total of $\tilde{O}(n^3)$ time.

The above approach differs from Algorithm I. However, random sampling is used in the selection algorithm of [56]. For a summary of these results see Table 4.

3.7 The de Bruijn Network

A directed de Bruijn network $DB(d, n)$ has $N = d^n$ nodes. A node v can be labeled as $d_n d_{n-1} \dots d_1$ where each d_i is a d -ary digit. Node $v = d_n d_{n-1} \dots d_1$ is connected to the nodes labeled $d_{n-1} \dots d_2 d_1 l$, denoted as $SH(v, l)$, where l is an arbitrary d -ary digit. Note that a rotation of n digits of v , denoted as $SH(v, d_n)$, represents an adjacent node of v . If v is also connected to the nodes labeled $l d_n d_{n-1} \dots d_2$, then the graph is referred to as undirected de Bruijn network. Hsu and Wei [21] have recently presented an $O(dn^2 \log d)$ time algorithm for sorting. If $d = 2$, their algorithm runs in time $2 \log^2 N$.

4 Selection Algorithms

The sequential selection algorithm of Blum, et. al. works as follows: 1) Partition the input of n numbers into groups with 5 elements in each group; 2) Find the median of each group; 3) Recursively compute the median M of

the group medians; 4) Partition the input into two using M as the splitter key. Part I has all the input keys $\leq M$ and Part II has the remaining keys. Identify the part that has the key to be selected and recursively perform an appropriate selection in this part.

One can easily show that the above algorithm runs in time $O(n)$. This is a good example of how deterministic sampling can be employed. A variant of the above has been used in all the deterministic parallel algorithms for selection.

Likewise, random sampling has been effectively applied to derive optimal or near optimal selections algorithms on various networks. A summary of such an algorithm is given below. To begin with all the input keys are *alive*. We are interested in selecting the i th smallest key.

Algorithm II

Step 1. Sample a set S of $o(n)$ keys at random from the collection X of alive keys.

Step 2. Sort the set S .

Step 3. Identify two keys l_1 and l_2 in S whose ranks in S are $i\frac{\delta}{n} - \delta$ and $i\frac{\delta}{n} + \delta$ respectively, δ being a ‘small’ integer.

(* The rank of l_1 in X is $< i$, and the rank of l_2 in X is $> i$, with high probability. *)

Step 4. Eliminate all the keys in X which are either $< l_1$ or $> l_2$.

Step 5. Repeat Steps 1 through 4 until the number of alive keys is ‘small’.

Step 6. Finally, concentrate and sort the alive keys.

Step 7. Perform an appropriate selection on the alive keys.

Next we enumerate known parallel selection algorithms on various models and show how the above two themes have been used repeatedly.

AUTHOR(S)	P	T	PROBLEM
Meggido [38]	n	$\tilde{O}(1)$	Maximal
Reischuk [58]	n	$\tilde{O}(1)$	Maximal
Rajasekaran and Sen [55]	n	$\tilde{O}(1)$	Maximal
Folklore	$\frac{n}{\log n}$	$\tilde{O}(\log n)$	General
Cole [11]	$\frac{n}{\log n \log^* n}$	$O(\log n \log^* n)$	General
Hagerup and Raman [17]	$\frac{n}{\log n}$	$O(\log n)$	General

Table 5: PRAM Selection

4.1 The PRAM

A straight forward implementation of Algorithm II on any of the PRAMs will yield an optimal randomized $\tilde{O}(\log n)$ time parallel algorithm for selection. On the CRCW PRAM, a similar algorithm can be used to solve the problem of finding the maximum of n given numbers in $\tilde{O}(1)$ time using n processors [55]. This result was proven in [38, 58] for the parallel comparison tree model. Cole used the idea of deterministic sampling to design an $O(\log n \log^* n)$ time $\frac{n}{\log n \log^* n}$ processor EREW PRAM algorithm [11]. The time bound of this algorithm has recently been improved to $O(\log n)$ using deterministic sampling as well as algorithms for approximate prefix computation [17].

4.2 The Mesh

The problem of selection where the number of processors is equal to the input size has been studied well by many researchers. The best known algorithm is due to Krizanc and Narayanan [27]. This randomized algorithm has a run time of $1.22\sqrt{n} + \tilde{o}(\sqrt{n})$ and is similar to Algorithm II. They also presented a deterministic algorithm for the case $n \neq p$ [28]. This algorithm had a run time of $O(\min\{p \log \frac{n}{p}, \max\{\frac{n}{p^{2/3}}, \sqrt{p}\}\})$. Rajasekaran, Chen, and Yooseph [51] have recently presented both deterministic and randomized algorithms for selection when $n \neq p$. Their deterministic algorithm has a run time of $O(\frac{n}{p} \log \log p + \sqrt{p} \log n)$ and the randomized algorithm resembles Algorithm II and runs in time $\tilde{O}((\frac{n}{p} + \sqrt{p}) \log \log p)$. A new deterministic selection scheme has been proposed in [51]. The idea is to employ the sequential algorithm of Blum et. al. [8] with some crucial modifications.

A summary of the selection scheme of [51] is given below since it can

be applied to any interconnection network to obtain good performance. To begin with, each one of the p processors has $\frac{n}{p}$ keys.

Algorithm III

$N := n$

Step 0. *if* $\log(n/p)$ is $\leq \log \log p$ *then*
 sort the elements at each node

else

 partition the keys at each node into $\log p$
 equal parts such that keys in one part will
 be \leq keys in parts to the right.

repeat

Step 1. In parallel find the median of keys at each
 node. Let M_q be the median and N_q be the number
 of remaining keys at node q , $1 \leq q \leq p$.

Step 2. Find the weighted median of M_1, M_2, \dots, M_p
 where key M_q has a weight of N_q , $1 \leq q \leq p$. Let
 M be the weighted median.

Step 3. Count the rank r_M of M from
 out of all the remaining keys.

Step 4. *if* $i \leq r_M$ *then*
 eliminate all the remaining keys that are $> M$
else

 eliminate all the remaining keys that are $\leq M$.

Step 5. Compute E , the number of keys eliminated.

if $i > r_M$ *then* $i := i - E$; $N := N - E$.

until $N \leq c$, c being a constant.

Output the i th smallest key from out of the remaining keys.

An implementation of the above algorithm on the Mesh yields the time bound of $O(\frac{n}{p} \log \log p + \sqrt{p} \log n)$. Table 6 summarizes known results.

4.3 Meshes with Buses

Here we consider the problem of selection when $n = p$. On a Mesh with reconfigurable buses, a lower bound of $\Omega(\log \log n)$ applies for deterministic selection, since selection even on the parallel comparison tree model has the same lower bound. ElGindy and Wegrowicz [14] applied an algorithm

AUTHOR(S)	YEAR	RUN TIME
Krizanc et. al. [27]	n	$1.22\sqrt{n} + \tilde{o}(\sqrt{n})$
Krizanc et. al. [28]	p	$O(\min\{p \log \frac{n}{p}, \max\{\frac{n}{p^{2/3}}, \sqrt{p}\}\})$
Rajasekaran et. al. [51]	p	$O(\frac{n}{p} \log \log p + \sqrt{p} \log n)$
Rajasekaran et. al. [51]	p	$\tilde{O}((\frac{n}{p} + \sqrt{p}) \log \log p)$

Table 6: Selection on the Mesh

AUTHOR(S)	YEAR	RUN TIME	DET./RAND.
ElGindy et. al. [14]	1991	$O(\log^2 n)$	DET.
Doctor et. al. [13]	1992	$\tilde{O}(\log^2 n)$	RAND.
Hao et. al. [18]	1992	$O(\log n)$	DET.
Rajasekaran [50]	1992	$\tilde{O}(\log^* n \log \log n)$	RAND.

Table 7: Mesh with Reconfigurable Buses: Selection

similar to that of [41] and showed that selection can be done on a p -node M_r in $O(\log^2 p)$ time. Followed by this, Doctor and Krizanc [13] presented a very simple randomized algorithm (similar to Algorithm II) that achieves the same time bound with high probability. This time bound was improved to $O(\log p)$ by Hao, McKenzie, and Stout [18]. Using an algorithm similar to that of Algorithm II and some other crucial properties of M_r , Rajasekaran [50] gave an $O(\log \log p \log^* p)$ expected time randomized algorithm. See Table 7.

On the other hand, $\Omega(p^{1/6})$ is a lower bound for selection on M_f [30]. A very nearly optimal algorithm has been given in [30]. An optimal randomized algorithm can be found in [50].

4.4 The Hypercube

A plethora of algorithms have been proposed for selection on the hypercube (both for the case $p = n$ and the case $p \neq n$). For the case $p = n$, an optimal $\tilde{O}(\log n)$ time randomized algorithm has been given in [57] and [48]. The algorithm in [57] is for sorting and hence can be applied for selection as well. On the other hand, the algorithm given in [48] is very simple.

AUTHOR(S)	YEAR	RUN TIME	DET./RAND.
Kumar et. al. [30]	1987	$O(n^{1/6}(\log n)^{2/3})$	DET.
Olariu et. al.	1992	$O(n^{1/6}(\log n)^{1/3})$	DET.
Rajasekaran [50]	1992	$\tilde{O}(n^{1/6})$	RAND.

Table 8: Selection on a Mesh with Fixed Buses

Model	Run Time	Lower Bound	Ref.
Sequential	$O(\frac{n}{p} \log \log p + \log^2 p \log(\frac{n}{p}))$	$\frac{n}{p} \log \log p + \log p$	[46]
Sequential	$\tilde{O}(\frac{n}{p} \log \log p + \log p \log \log p)$	$\frac{n}{p} \log \log p + \log p$	[48]
Sequential	$O(\log^* n \log n)$	$\log n$	[6]
Sequential	$\tilde{O}(\log n)$	$\log n$	[57, 48]
Weak Parallel	$O(\frac{n}{p} + \log p \log \log p)$	$\frac{n}{p} + \log p$	[46]
Weak Parallel	$\tilde{O}(\frac{n}{p} + \log p)$	$\frac{n}{p} + \log p$	[48]

Table 9: Selection on the Hypercube

[48]’s algorithm has been implemented on CM-2 and empirical results are promising [51]. The best known deterministic algorithm is due to Berthomé, et. al. [6] and has a run time of $O(\log n \log^* n)$.

For the case of $p \neq n$, refer to Table 9 for a summary of the best known algorithms. All the algorithms in this Table use the technique of sampling (either deterministic or randomized). A slightly better deterministic algorithm can be obtained using Algorithm III as has been shown in [51]. The run time will be $O(\frac{n}{p} \log \log p + \log^2 p \log \log p)$. If a better sorting algorithm is discovered for the hypercube, this time bound will improve further.

4.5 The Star Graph

The only known selection algorithm on the star graph is due to Rajasekaran and Wei [56]. This randomized algorithm runs in time $\tilde{O}(n^2)$ on an $n!$ -node star graph. Within the same asymptotic time bound, this algorithm can perform n different selections. A sorting algorithm with a run time of $\tilde{O}(n^3)$ follows from this algorithm.

5 Conclusions and Open Problems

In this paper we have surveyed known parallel algorithms for sorting and selection on various interconnection networks. We have also identified some very commonly used techniques for the design of such algorithms. Here is a partial list of open problems in this exciting area of algorithmic research: 1) Is there a simple $2\sqrt{n} - 2$ step sorting algorithm for the Mesh with small queues? 2) What is the best run time achievable for selection on the Mesh (when $p = n$)? 3) Is there an $O(\log \log n)$ time algorithm for selection on M_r (either randomized or deterministic)? 4) Can one design a simple deterministic $O(\log n)$ time sorting algorithm for the hypercube? 5) Can the best known sorting times for the star graph and the de Bruijn graph be improved?

References

- [1] A.V. Aho, J.E. Hopcroft, and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley Publishing Company, 1974.
- [2] M. Ajtai, J. Komlós and E. Szemerédi, An $O(n \log n)$ Sorting Network, Proc. 15th ACM Symposium on Theory of Computing, 1983, pp. 1-9.
- [3] K.E. Batcher, Sorting Networks and their Applications, Proc. 1968 Spring Joint Computer Conference, vol. 32, AFIPS Press, 1968, pp. 307-314.
- [4] P. Beame and J. Hastad, Optimal Bounds for Decision Problems on the CRCW PRAM, Proc. 19th ACM Symposium on Theory Of Computing, 1987, pp. 83-93.
- [5] Y. Ben-Asher, D. Peleg, R. Ramaswami, and A. Schuster, The Power of Reconfiguration, Journal of Parallel and Distributed Computing, 1991, pp. 139-153.
- [6] P. Berthomé, A. Ferreira, B.M. Maggs, S. Perennes, and C.G. Plaxton, Sorting-Based Selection Algorithms for Hypercubic Networks, Proc. International Parallel Processing Symposium, 1993, pp. 89-95.
- [7] G.E. Blelloch, C.E. Leiserson, B.M. Maggs, C.G. Plaxton, S.J. Smith, M. Zagher, A Comparison of Sorting Algorithms for the Connection Ma-

- chine CM-2, Proc. 3rd Annual ACM Symposium on Parallel Algorithms and Architectures, 1991.
- [8] M. Blum, R. Floyd, V.R. Pratt, R. Rivest, and R. Tarjan, Time Bounds for Selection, *Journal of Computer and System Science*, 7(4), 1972, pp. 448-461.
- [9] J.C. Cogolludo and S. Rajasekaran, Permutation Routing on Reconfigurable Meshes, Proc. Fourth Annual International Symposium on Algorithms and Computation, Springer-Verlag Lecture Notes in Computer Science 762, 1993, pp. 157-166.
- [10] R. Cole, Parallel Merge Sort, *SIAM Journal on Computing*, vol. 17, no. 4, 1988, pp. 770-785.
- [11] R. Cole, An Optimally Efficient Selection Algorithm, *Information Processing Letters* 26, Jan. 1988, pp. 295-299.
- [12] R.E. Cypher and C.G. Plaxton, Deterministic Sorting in Nearly Logarithmic Time on the Hypercube and Related Computers, Proc. ACM Symposium on Theory of Computing, 1990, pp. 193-203.
- [13] D.P. Doctor and D. Krizanc, Three Algorithms for Selection on the Reconfigurable Mesh, Technical Report TR-219, School of Computer Science, Carleton University, February 1993.
- [14] H. ElGindy and P. Wegrowicz, Selection on the Reconfigurable Mesh, Proc. International Conference on Parallel Processing, 1991, Vol. III, pp. 26-33.
- [15] R.W. Floyd, and R.L. Rivest, Expected Time Bounds for Selection, *Communications of the ACM*, Vol. 18, No.3, 1975, pp. 165-172.
- [16] W.D. Frazer and A.C. McKellar, Samplesort: A Sampling Approach to Minimal Storage Tree Sorting, *Journal of the ACM*, vol.17, no.3, 1970, pp. 496-507.
- [17] T. Hagerup and R. Raman, An Optimal Parallel Algorithm for Selection, Proc. 5th Annual ACM Symposium on Parallel Algorithms and Architectures, 1993.
- [18] E. Hao, P.D. McKenzie and Q.F. Stout, Selection on the Reconfigurable Mesh, Proc. Frontiers of Massively Parallel Computation, 1992, pp. 38-45.

- [19] W.L. Hightower, J.F. Prins, J.H. Reif, Implementation of Randomized Sorting on Large Parallel Machines, Proc. 4th Annual ACM Symposium on Parallel Algorithms and Architectures, 1992, pp. 158-167.
- [20] C.A.R. Hoare, Quicksort, Computer Journal 5, 1962, pp. 10-15.
- [21] D.F. Hsu, D.S.L. Wei, Permutation Routing and Sorting on Directed de Bruijn Networks, Technical Report, University of Aizu, Japan, 1994.
- [22] J. Jang, H. Park, and V.K. Prasanna, A Fast Algorithm for Computing Histograms on a Reconfigurable Mesh, Proc. Frontiers of Massively Parallel Computing, 1992, pp. 244-251.
- [23] J. Jang and V.K. Prasanna, An Optimal Sorting Algorithm on Reconfigurable Mesh, Proc. International Parallel Processing Symposium, 1992, pp. 130-137.
- [24] C. Kaklamanis and D. Krizanc, Optimal Sorting on Mesh-Connected Processor Arrays, Proc. 4th Annual ACM Symposium on Parallel Algorithms and Architectures, 1992, pp. 50-59.
- [25] C. Kaklamanis, D. Krizanc, L. Narayanan, and Th. Tsantilas, Randomized Sorting and Selection on Mesh Connected Processor Arrays, Proc. 3rd Annual ACM Symposium on Parallel Algorithms and Architectures, 1991.
- [26] M. Kaufmann, S. Torsten, and J. Sibeyn, Derandomizing Algorithms for Routing and Sorting on Meshes, Proc. 5th Annual ACM-SIAM Symposium on Discrete Algorithms, 1994, pp. 669-679.
- [27] D. Krizanc, and L. Narayanan, Optimal Algorithms for Selection on a Mesh-Connected Processor Array, Proc. Symposium on Parallel and Distributed Processing, 1992.
- [28] D. Krizanc and L. Narayanan, Multi-packet Selection on a Mesh-Connected Processor Array, Proc. International Parallel Processing Symposium, 1992.
- [29] D. Krizanc, S. Rajasekaran, and S. Shende, A Comparison of Meshes with Static Buses and Half-Duplex, to appear in Parallel Processing Letters, 1995.

- [30] V.K.P. Kumar and C.S. Raghavendra, Array Processor with Multiple Broadcasting, *Journal of Parallel and Distributed Computing* 4, 1987, pp. 173-190.
- [31] M. Kunde, Concentrated Regular Data Streams on Grids: Sorting and Routing Near to the Bisection Bound, *Proc. IEEE Symposium on Foundations of Computer Science*, 1991, pp. 141-150.
- [32] M. Kunde, Block Gossiping on Grids and Tori: Sorting and Routing Match the Bisection Bound Deterministically, *Proc. European Symposium on Algorithms*, 1993, pp. 272-283.
- [33] T. Leighton, Tight Bounds on the Complexity of Parallel Sorting, *IEEE Transactions on Computers*, vol. C34 (4), April 1985, pp. 344-354.
- [34] T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays-Trees-Hypercube*, Morgan-Kaufmann Publishers, 1992.
- [35] H. Li and Q. Stout, *Reconfigurable Massively Parallel Computers*, Prentice-Hall Publishers, 1991.
- [36] R. Lin, S. Olariu, J. Schwing, and J. Zhang, A VLSI-optimal Constant Time Sorting on Reconfigurable Mesh, *Proc. European Workshop on Parallel Computing*, 1992.
- [37] Y. Ma, S. Sen, and D. Scherson, The Distance Bound for Sorting on Mesh Connected Processor Arrays is Tight, *Proc. IEEE Symposium on Foundations of Computer Science*, 1986, pp. 255-263.
- [38] N. Meggido, Parallel Algorithms for Finding the Maximum and the Median Almost Surely in Constant Time, Preliminary Report, CS Department, Carnegie-Mellon University, Pittsburg, PA, Oct. 1982.
- [39] A. Menn and A.K. Somani, An Efficient Sorting Algorithm for the Star Graph Interconnection Network, *Proc. International Conference on Parallel Processing*, 1990, vol. 3, pp. 1-8.
- [40] O. Menzilcioglu, H.T. Kung, and S.W. Song, Comprehensive Evaluation of a Two-Dimensional Configurable Array, *Proc. 19th Symposium on Fault Tolerant Computing*, 1989, pp. 93-100.
- [41] J.I. Munro and M.S. Paterson, Selection and Sorting with Limited Storage, *Theoretical Computer Science* 12, 1980, pp. 315-323.

- [42] K. Nakano, D. Peleg, and A. Schuster, Constant-time Sorting on a Reconfigurable Mesh, Manuscript, 1992.
- [43] D. Nassimi and S. Sahni, Parallel Permutation and Sorting Algorithms and a New Generalized Connection Network, *Journal of the ACM*, 29(3), 1982, pp. 642-667.
- [44] M. Nigam and S. Sahni, Sorting n^2 Numbers on $n \times n$ Meshes, *Proc. International Parallel Processing Symposium*, 1993, pp. 73-78.
- [45] M. Nigam and S. Sahni, Sorting n Numbers on $n \times n$ Reconfigurable Meshes with Buses, *Proc. International Parallel Processing Symposium*, 1993, pp. 174-181.
- [46] C.G. Plaxton, Efficient Computation on Sparse Interconnection Networks, Ph. D. Thesis, Department of Computer Science, Stanford University, 1989.
- [47] F. Preparata, New Parallel Sorting Schemes, *IEEE Transactions on Computers*, vol. C27, no. 7, 1978, pp. 669-673.
- [48] S. Rajasekaran, Randomized Parallel Selection, *Proc. Symposium on Foundations of Software Technology and Theoretical Computer Science*, 1990, pp. 215-224.
- [49] S. Rajasekaran, $k - k$ Routing, $k - k$ Sorting, and Cut Through Routing on the Mesh, to appear in *Journal of Algorithms*, 1995.
- [50] S. Rajasekaran, Mesh Connected Computers with Fixed and Reconfigurable Buses: Packet Routing, Sorting, and Selection, *Proc. First Annual European Symposium on Algorithms*, Springer-Verlag Lecture Notes in Computer Science 726, 1993, pp. 309-320.
- [51] S. Rajasekaran, W. Chen, and S. Yooseph, Unifying Themes for Parallel Selection, *Proc. Fourth International Symposium on Algorithms and Computation (ISAAC)*, Springer-Verlag Lecture Notes in Computer Science 834, 1994, pp. 92-100.
- [52] S. Rajasekaran and T. McKendall, Permutation Routing and Sorting on the Reconfigurable Mesh, Technical Report MS-CIS-92-36, Department of Computer and Information Science, University of Pennsylvania, May 1992.

- [53] S. Rajasekaran and J.H. Reif, Optimal and Sub-Logarithmic Time Randomized Parallel Sorting Algorithms, *SIAM Journal on Computing*, vol. 18, no. 3, 1989, pp. 594-607.
- [54] S. Rajasekaran and J.H. Reif, Derivation of Randomized Sorting and Selection Algorithms, in *Parallel Algorithm Derivation and Program Transformation*, Edited by R. Paige, J.H. Reif, and R. Wachter, Kluwer Academic Publishers, 1993, pp. 187-205.
- [55] S. Rajasekaran, and S. Sen, Random Sampling Techniques and Parallel Algorithms Design, in *Synthesis of Parallel Algorithms*, Editor: J.H. Reif, Morgan-Kaufman Publishers, 1993, pp. 411-451.
- [56] S. Rajasekaran and D.S.L. Wei, Selection, Routing, and Sorting on the Star Graph, *Proc. International Parallel Processing Symposium*, pp. 661-665, April 1993.
- [57] J.H. Reif and L.G. Valiant, A Logarithmic Time Sort for Linear Size Networks, *Journal of the ACM*, 34(1), Jan. 1987, pp. 60-76.
- [58] R. Reischuk, Probabilistic Parallel Algorithms for Sorting and Selection, *SIAM Journal of Computing*, Vol. 14, No. 2, 1985, pp. 396-409.
- [59] C. Schnorr and A. Shamir, An Optimal Sorting Algorithm for Mesh-Connected Computers, *Proc. ACM Symposium on Theory of Computing*, 1986, pp. 255-263.
- [60] J.F. Sibeyn, M. Kaufmann, and R. Raman, Randomized Routing on Meshes with Buses, *Proc. European Symposium on Algorithms*, 1993, pp. 333-344.
- [61] T.M. Stricker, Supporting the Hypercube Programming Model on Mesh Architectures (A Fast Sorter for iWarp Tori), *Proc. 4th Annual ACM Symposium on Parallel Algorithms and Architectures*, 1992, pp. 148-157.
- [62] X. Thibault, D. Comte, and P. Siron, A Reconfigurable Optical Interconnection Network for Highly Parallel Architecture, *Proc. Symposium on the Frontiers of Massively Parallel Computation*, 1989.
- [63] C.D. Thompson and H.T. Kung, Sorting on a Mesh Connected Parallel Computer, *Communications of the ACM*, vol. 20, no.4, 1977, pp. 263-271.

- [64] B.F. Wang, G.H. Chen, and F.C. Lin, Constant Time Sorting on a Processor Array with a Reconfigurable Bus System, Information Processing Letters, 34(4), April 1990.

APPENDIX A: Chernoff Bounds

Lemma A.1 *If X is binomial with parameters (n, p) , and $m > np$ is an integer, then*

$$\text{Probability}(X \geq m) \leq \left(\frac{np}{m}\right)^m e^{m-np}. \quad (1)$$

Also,

$$\text{Probability}(X \leq \lfloor (1 - \epsilon)pn \rfloor) \leq \exp(-\epsilon^2 np/2) \quad (2)$$

and

$$\text{Probability}(X \geq \lceil (1 + \epsilon)np \rceil) \leq \exp(-\epsilon^2 np/3) \quad (3)$$

for all $0 < \epsilon < 1$.

DEPARTMENT OF COMPUTER AND INFORMATION SCIENCE,
UNIVERSITY OF FLORIDA, GAINESVILLE, FL 32611
E-mail address: raj@cis.ufl.edu