Chapter 1

RANDOM SAMPLING: SORTING AND SELECTION

Danny Krizanc
Dept. of Math. and Comp. Sci.
Wesleyan University
dkrizanc@wesleyan.edu

Sanguthevar Rajasekaran
Dept. of Comp. and Info. Sci. and Engg.
University of Florida
raj@cise.ufl.edu

Abstract Random sampling techniques have played a vital role in the design of sorting and selection algorithms for numerous models of computing. In this article we provide a summary of sorting and selection algorithms that have been devised using random sampling. Models of computations treated include the parallel comparison tree, the PRAM, the mesh, the mesh with fixed, reconfigurable, and optical buses, the hypercube family, and parallel disk systems.

1. INTRODUCTION

Comparison problems such as sorting and selection have been studied by researchers extensively owing to their paramount importance. Given a sequence of n keys the problem of sorting is to rearrange this sequence in nondecreasing order. The selection problem takes as input a sequence of n keys and an integer i ($1 \leq i \leq n$). The problem is to identify the ith smallest key of the sequence.

Optimal (comparison based) sequential RAM algorithms are known for sorting and selection. Sorting algorithms such as mergesort, heapsort, etc. run in time $O(n \log n)$ time in the worst case (see e.g., [3, 30]). The selection algorithm of Blum et al. [15] runs in linear time.

Optimal or near-optimal algorithms for sorting and selection have been developed for numerous other models of computing as well. The technique of random sampling has been successfully employed in many of these algorithms.

In this article we provide a survey of some of these algorithms given for parallel models of computing.

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

1.1 AN INTRODUCTION TO MODELS OF COMPUTING

In this section we give a brief introduction to the models of computing considered in this article.

1.1.1 The Parallel Comparison Tree. The Parallel Comparison Tree (PCT) model [84] is the natural generalization of the sequential comparison tree model [3] to the parallel setting. The basic operation available to processors is the comparison of two keys. With p processors, p comparisons may be performed simultaneously in one step. Depending on which of the 2^p possible results is attained, the next set of p comparisons is chosen. The computation ends when sufficient information is discovered about the relationships of the keys to specify the solution to the given problem. The deterministic complexity of a problem in this model is the number of steps required for the worst case input or the minimum depth of a tree solving the problem, as a function of the size of the input sequence and the number of processors used.

We note that in this model we do not consider any of the overheads, such as processor communication, memory accesses, etc., associated with performing the comparisons and making the appropriate deductions from the results of the comparisons. However, in cases where the cost of comparisons dominates the computation, upper bounds in this model can often be translated into upper bounds in more restricted models and in all cases where algorithms base their decisions solely upon comparisons, lower bounds in this model translate to lower bounds in these other models.

The model is easily extended to allow random computations. In the randomized PCT model, at each step we introduce a probability distribution over the choice of which p comparisons are to be performed. In this case, the complexity is the expected number of steps required on the worst case input.

1.1.2 The Parallel Random Access Machine. The Parallel Random Access Machine (PRAM) is the natural generalization of the RAM model to the parallel setting. In it, p synchronous processors, each identical to a RAM, communicate through the use of a shared memory. There are three main variants of the PRAM depending on what restrictions are placed on concurrent access to the same memory cell in the shared memory. The Exclusive Read Exclusive Write (EREW) PRAM does not allow any simultaneous access to the same memory cell. The Concurrent Read Exclusive Write (CREW) PRAM allows concurrent reads to take place but does not allow concurrent writes. The Concurrent Read Concurrent Write (CRCW) PRAM allows both concurrent reads and concurrent writes. There are three standard varieties of CRCW PRAM depending on the interpretation of a concurrent write operation. In the common CRCW PRAM, it is required that concurrent writes to a cell are all writing the same value. In the arbitrary CRCW PRAM model an arbitrary value among those being written is chosen. In the priority CRCW PRAM the value written by the lowest indexed processor is the result of the concurrent write. It is easy to see that the CRCW PRAM is stronger than the CREW PRAM which in turn is stronger than the EREW PRAM. It is easy to show that they are all three logarithmically related.

1.1.3 The Mesh. A mesh is a $\sqrt{p} \times \sqrt{p}$ square grid where there is a processor at each grid point. Every processor is connected to its four or less neighbors through bidirectional links. Each processor can communicate with all of its neighbors in one unit of time.

1.1.4 Mesh with Buses. Two variants of the mesh assume the existence of electrical communication buses: 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 each row and each column has an associated broadcast bus. A bus can be used to broadcast a message in every time step. It is assumed that the message broadcast along a bus can be read by all the processors connected to this bus in the same time unit.

M_r also employs buses but these buses are reconfigurable. Reconfigurability of the buses is achieved as follows. Each processor has (at most) four switches, one for each of its neighbors. This switch can be dynamically set on or off. If a switch of a processor is on, it means that the processor is connected to the corresponding neighbor. Depending on how the switches of the processors are set, we can get several disjoint buses. For instance we can form a row bus by setting all the switches along the row on. Each bus functions similar to the buses of M_f, i.e., a message can be broadcast in any bus at every time step and this message can be read by all the processors connected to the bus in the same time step.

In M_f and M_r we assume the existence of electrical buses. Optical technology can be employed to realize these buses in which case we get meshes with optical buses. Several such models have been investigated in the literature.

1.1.5 The Hypercube. A hypercube of dimension ℓ has $p = 2^\ell$ nodes and $\ell 2^{\ell-1}$ edges. Each node in an ℓ-dimensional hypercube can be labelled with an ℓ-bit binary number. Nodes x and y in a hypercube will be connected by a bidirectional link if and only if x and y (considered as binary numbers) differ in exactly one bit position. Thus there are exactly ℓ edges going out of (and coming into) any vertex.

If a hypercube processor can communicate with only one neighbor at any time step, this version of the hypercube will be called the *sequential model*. If a processor can communicate with all its neighbors in a time step then this variant of the hypercube is called the *parallel model*.

A multitude of other important networks are related to the hypercube. Among the more important (constant degree) members of the hypercube family are the Cube Connected Cycle (CCC), the Butterfly and the de Bruijn family of networks. For definitions of these networks see [46].

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

1.1.7 Parallel Disk Systems. With the widening gap between processor speeds and disk access speeds, the I/O bottleneck has become critical. Parallel Disk Systems (PDS) have been introduced to alleviate this bottleneck [85]. In this model there are D distinct and independent disk drives. The disks can simultaneously transmit a block of data. A block consists of B records. If M is the internal memory size, then one usually requires that $M \geq 2DB$. While analyzing algorithms developed for this model, one typically computes the number of I/O operations needed for the algorithm. Local computations are neglected since the time for I/O is much more than the time for local computations.

2. RANDOM SAMPLING

Random sampling has been employed in the development of numerous sorting and selection algorithms. One of the early papers that dealt with sampling was due to Frazer and McKellar [25]. They proposed the following sorting algorithm which can be thought of as a generalization of the quicksort algorithm [29]: 1) Sample $o(n)$ keys from the input and use any (possibly nonoptimal) algorithm to sort them; 2) Use these sample keys to partition the input into subsequences; and 3) Sort each subsequence independently. Sorting algorithms for several models of computing have been designed using this technique.

Random sampling has also dominated the arena of selection algorithms. As an example, Floyd and Rivest [24] proposed the following scheme for selection:

1) Randomly pick $o(n)$ keys from the input and identify two keys ℓ_1 and ℓ_2 from this sample such that the element to be selected has a value in the range $[\ell_1, \ell_2]$ and not many input keys are in the range $[\ell_1, \ell_2]$; 2) Delete all the input keys that are outside the range $[\ell_1, \ell_2]$; and 3) Perform an appropriate selection from out of the remaining keys. The number of comparisons made by this algorithm to identify the ith smallest key is $n + \min\{i, n-i\} + o(n)$ with high probability. The proof of this fact and related sampling bounds are generally encapsulated in a "sampling lemma."

2.1 A SAMPLING LEMMA

Several sampling lemmas have been proven in the literature. One of the basic lemmas deals with the following sampling process. Let $S = \{k_1, k_2, \ldots, k_s\}$ be a random sample from a sequence X of n numbers. Let the sorted order of S be $k'_1, k'_2, \ldots, k'_s$. If r_i is the rank of k'_i in X, many algorithms benefit from 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.) A proof of the following Lemma can be found in [74].

Lemma 2..1 *For every α, Prob.* $\left(|r_i - i\frac{n}{s}| > \sqrt{3\alpha}\frac{n}{\sqrt{s}}\sqrt{\log n}\right) < n^{-\alpha}$.

Notation. We say a randomized algorithm has a resource bound of $\widetilde{O}(f(n))$ if there exists a constant c such that the amount of resource used is no more than $c\alpha f(n)$ **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 $\widetilde{o}(.)$, $\widetilde{\Omega}(.)$, etc.

2.2 ORGANIZATION OF THIS PAPER

The rest of this article is organized as follows. Sections 3 and 4 are devoted to sorting and selection problems, respectively. In Section 5 we provide some concluding remarks.

3. SAMPLING BASED SORTING

3.1 A GENERAL THEME

The following is an idea introduced by Frazer and McKellar [25] that has been implemented over a variety of models.

Algorithm I

Step 1. Pick a random sample of n^ϵ (for some constant $\epsilon < 1$) input 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.

It is easy to show using lemma 2..1 that the splitter keys very evenly distribute the keys so that in the last step the work done by each processor is approximately the same, with high probability.

3.2 THE PCT

One of the classical results in parallel sorting is Batcher's algorithm [10]. This algorithm is based on the idea of bitonic sorting and was proposed for the hypercube and hence can be run on stronger models such as any of the PRAMs and the PCT as well. Batcher's algorithm runs in $O(\log^2 n)$ time when sorting n keys using n processors. Followed by this, a very nearly optimal algorithm was given by Preparata [62]. 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 [78] which implies the result for the randomized PCT. At around the same time Ajtai, Komlós, and Szemerédi announced their sorting network of depth $O(\log n)$ [4]. This established the upper bound for sorting on the PCT for the case $p \leq n$. This was extended by Alon, Azar and Vishkin [7] to the case $p > n$. Taken together their results show the complexity of sorting n keys on a p processor PCT is $\Theta(\log n / \log(1 + p/n))$. The matching lower bound for randomized or deterministic sorting was first shown by Alon and Azar [6]. A significantly simpler proof of the same result was provided by Boppana [16].

3.3 THE PRAM

As was stated above, Reischuk was the first to design an optimal logarithmic time a randomized CREW PRAM algorithm for sorting [78]. His algorithm may be derived from Algorithm I. The AKS sorting circuit [4] implies the existence of a deterministic EREW PRAM algorithm running in logarithmic time. 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 [45]. Though several attempts have been made to improve the constant in the time bound, the algorithm of [4] remains a result of only theoretical interest.

In 1987 Cole presented an optimal logarithmic time EREW PRAM algorithm for sorting, the constant in the time bound being reasonably small [18]. 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 Beame 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 [11]. Rajasekaran and Reif [73] were able to obtain a randomized algorithm for sorting on the CRCW PRAM that runs in time $\widetilde{O}(\frac{\log n}{\log\log n})$, 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.

3.4 THE MESH

The first asymptotically optimal sorting algorithm for the mesh was given by Thompson and Kung [83]. 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 $2\sqrt{n}-2$, [83]'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 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 [79]. 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 [49] gave a near optimal algorithm for a related model. Kaklamanis et al. presented a very interesting algorithm for sorting with a run time of $2.5\sqrt{n} + \tilde{o}(\sqrt{n})$ [35]. This algorithm was randomized and used queues of size $O(1)$. The underlying idea here is the same as that of Algorithm I. Kaklamanis and Krizanc later improved this time bound to $2\sqrt{n} + \tilde{o}(\sqrt{n})$ [34].

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 [43]; Nigam and Sahni have given a $(2 + \epsilon)\sqrt{n} + o(\sqrt{n})$ time algorithm (for any fixed $\epsilon > 0$) [56]; Also Kaufmann, Sibeyn, and Torsten have offered a $2\sqrt{n} + o(\sqrt{n})$ time algorithm [36]. The third algorithm closely resembles the one given by [34] 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 [65]. Kunde [44] has matched this result with a deterministic algorithm.

3.5 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., [66]). 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 $\widetilde{o}(\sqrt{n})$ more than the time needed for packet routing under the same settings as has been proven in [72]. This randomized algorithm is also similar to Algorithm I. In [41], 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 $\frac{17}{18}\sqrt{n} + \widetilde{o}(\sqrt{n})$ [17]. For M_f, the best known packet routing time is $0.79\sqrt{n} + \widetilde{o}(\sqrt{n})$ [80]. Therefore, sorting can be done on M_r in time $\frac{17}{18}\sqrt{n} + \widetilde{o}(\sqrt{n})$ and on M_f in time $0.79\sqrt{n} + \widetilde{o}(\sqrt{n})$.

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 number of processors [11]. A constant time algorithm using n^3 processors appears in [86]. The processor bound was improved to n^2 in independent works [33], [48], [54], [57].

3.6 THE HYPERCUBE

Batcher's algorithm runs in $O(\log^2 n)$ time on an n-node hypercube [10]. 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 [55] 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 [77]. 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 [21]. 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 [46]. Hsu and Wei [31] have recently presented an $O(dn^2 \log d)$ time algorithm for sorting on the $d^n = N$ node de Bruijn network. If $d = 2$, their algorithm runs in time $2\log^2 N$ steps.

3.7 THE STAR GRAPH

Menn and Somani [51] employed an algorithm similar to that of Schnorr and Shamir [79] to show that sorting can be done on a star graph with $n!$ nodes in $O(n^3 \log n)$ time. Rajasekaran and Wei [76] have offered a randomized algorithm with a time bound of $\widetilde{O}(n^3)$. This algorithm is based on a randomized selection algorithm that they derive. A summary of this algorithm follows:

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 ℓth phase, each key will be in its right $S_{n-\ell}$ (for $1 \leq \ell \leq n-1$). Selection in each phase takes $\widetilde{O}(n^2)$ 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 $\widetilde{O}(n)$ time. Thus each phase takes $\widetilde{O}(n^2)$ time, accounting for a total of $\widetilde{O}(n^3)$ time.

The above approach differs from Algorithm I. However, random sampling is used in the selection algorithm of [76].

3.8 PARALLEL DISK SYSTEMS

The problem of disk sorting was first studied by Aggarwal and Vitter in their fundamental paper [2]. In the model they considered, each I/O operation results in the transfer of D blocks each block having B records. A more realistic model was envisioned in [85]. Several asymptotically optimal algorithms have been given for sorting on this model. Nodine and Vitter's optimal algorithm [58] involves solving certain matching problems. Aggarwal and Plaxton's optimal algorithm [1] is based on the Sharesort algorithm of Cypher and Plaxton. Vitter and Shriver gave an optimal randomized algorithm for disk sorting [85]. All these results are highly nontrivial and theoretically interesting. However, the underlying constants in their time bounds are high.

In practice the simple disk-striped mergesort (DSM) is used [9], even though it is not asymptotically optimal. DSM has the advantages of simplicity and a small constant. Data accesses made by DSM is such that at any I/O operation, the same portions of the D disks are accessed. This has the effect of having a single disk which can transfer DB records in a single I/O operation. An $\frac{M}{DB}$-way mergesort is employed by this algorithm. To start with, initial runs are formed in one pass through the data. At the end the disk has N/M runs each of length M. Next, $\frac{M}{DB}$ runs are merged at a time. Blocks of any run are uniformly striped across the disks so that in future they can be accessed in parallel utilizing the full bandwidth. Each phase of merging involves one pass through the data. There are $\frac{\log(N/M)}{\log(M/DB)}$ phases and hence the total number of passes made by DSM is $\frac{\log(N/M)}{\log(M/DB)}$. In other words, the total number of I/O read operations performed by the algorithm is $\frac{N}{DB}\left(1+\frac{\log(N/M)}{\log(M/DB)}\right)$. The constant here is just 1.

A known lower bound on the number of passes for parallel disk sorting is $\Omega\left(\frac{\log(N/B)}{\log(M/B)}\right)$. Here N is the input size, M is the core memory size, and B is the block size. If one assumes that N is a polynomial in M and that B is small (which are readily satisfied in practice), the lower bound simply yields $\Omega(1)$ passes. A number of optimal algorithms that make only $O(1)$ passes have been proposed in the literature. So, the challenge in the design of parallel disk sorting algorithms is in reducing this constant. If $M = 2DB$, the number of passes made by DSM is $1 + \log(N/M)$, which indeed can be very high.

Recently, much work has been done that deals with the practical aspects of parallel disk systems. Pai, Schaffer, and Varman [59] analyzed the average case performance of a simple merging algorithm, employing an approximate model of average case inputs. Barve, Grove, and Vitter [9] have presented a simple randomized algorithm (SRM) and analyzed its performance. The analysis involves the solution of certain occupancy problems. The expected number R_{SRM} of I/O read operations made by their algorithm is such that

$$R_{SRM} \leq \frac{N}{DB}\left[1+\frac{\ln(N/M)}{\ln kD}\frac{\ln D}{k\ln\ln D}\left(1+\frac{\ln\ln\ln D}{\ln\ln D}+\frac{1+\ln k}{\ln\ln D}+O(1)\right)\right] \quad (1.1)$$

The algorithm merges $R = kD$ runs at a time, for some integer k. When $R = \Omega(D\log D)$, the expected performance of their algorithm is optimal. However, in this case, the internal memory needed is $\Omega(BD\log D)$. They have also compared SRM with DSM through simulations and shown that SRM performs better than DSM.

In a recent work, Rajasekaran [67] has presented a simple algorithm (called (ℓ, m)-merge sort (LMM)) that is asymptotically optimal (under the assumptions that N is a polynomial in M and B is small) and the underlying constant is small. LMM is as simple as the DSM. LMM makes less number of passes through the data than DSM when D is large. Recent implementation results [60] [71] indicate that LMM is competitive in practice.

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 selection algorithms in various parallel models. A summary of such an algorithm is given below. To begin with all the input keys are *alive*. We are interested in selecting the ith 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 $i\frac{s}{n} - \delta$ and $i\frac{s}{n} + \delta$ respectively, δ being a 'small' integer.
(* Using lemma 2..1 or a variant it is easy to show 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 theme has been used repeatedly.

4.1 THE PCT

Valiant [84] showed a deterministic lower bound for selection on the PCT. A deterministic upper bound was shown by Azar and Pippenger [8] (building on the work of Ajtai et al. [5]). Their results together show that deterministic selection from a sequence of n keys using p processors requires $\Theta(n/p + \log(\log n/\log(2 + p/n)))$ steps.

Meggido [50] and independently Reischuk [78] showed that the above lower bound could be "beaten" using randomization by providing an optimal randomized PCT algorithm for selection that runs in $\Theta(n/p + 1)$ steps. Both of their algorithms are implementations of Algorithm II on the randomized PCT.

The results above show that there is a significant gap between the randomized and deterministic parallel complexity of selection in the PCT model. A partial explanation of this phenomenon is given in [37] where a tight tradeoff between the amount of randomness used by a randomized PCT for selection and its performance, measured by the time it requires to complete its computation with a given failure probability, is shown.

4.2 THE PRAM

A straight forward implementation of Algorithm II on any of the PRAMs will yield an optimal randomized $\widetilde{O}(\log n)$ 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 $\widetilde{O}(1)$ time using n processors [75]. 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 [19]. The time bound of this algorithm has been improved to $O(\log n)$ using deterministic sampling as well as algorithms for approximate prefix computation [26].

4.3 THE MESH

The problem of selection on the mesh where the number of processors is equal to the input size has been studied by many researchers. The best known algorithm is due to Condon and Narayanan [20]. This randomized algorithm has a run time of $1.15\sqrt{n} + \widetilde{o}(\sqrt{n})$ and is similar to Algorithm II. The best known deterministic algorithm has a run time of $1.44\sqrt{n}+o(\sqrt{n})$ [40]. Krizanc and Narayanan provide optimal (to within an additive term) $\sqrt{n} + o(\sqrt{n})$ step algorithms for certain special cases of selection, e.g., the maximum [38]. For the case $n > p$ Krizanc and Narayanan [39], present a deterministic algorithm with a run time of $O(\min\{p \log \frac{n}{p}, \max\{\frac{n}{p^{2/3}}, \sqrt{p}\}\})$. Rajasekaran, Chen, and Yooseph [70] have presented both deterministic and randomized algorithms for selection when $n > p$. 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 $\widetilde{O}((\frac{n}{p}+\sqrt{p}) \log\log p)$. A new deterministic selection scheme has been proposed in [70]. The idea is to employ the sequential algorithm of Blum et. al. [15] with some crucial modifications.

A summary of the selection scheme of [70] 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 $M_1, M_2, \ldots, M_p$
    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.
```

When the above algorithm is implemented on the mesh the resulting time bound is $O(\frac{n}{p}\log\log p + \sqrt{p}\log n)$.

4.4 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 comparison based deterministic selection, since selection even on the parallel comparison tree model has the same lower bound. ElGindy and Wegrowicz [23] applied an algorithm similar to that of [53] 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 [22] 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 [27]. Using an algorithm similar to that of Algorithm II and some other crucial properties of

M_r, Rajasekaran [64, 69] gave an $O(\log\log p \log^* p)$ expected time randomized algorithm.

On the other hand, $\Omega(p^{1/6})$ is a lower bound for selection on M_f [42]. A very nearly optimal algorithm has been given in [42]. An optimal randomized algorithm can be found in [64, 69].

4.5 THE HYPERCUBE

A plethora of algorithms have been proposed for selection on the hypercube (both for the case $p = n$ and the case $p < n$). For the case $p = n$, an optimal $\widetilde{O}(\log n)$ time randomized algorithm has been given in [77] and [63]. The algorithm in [77] is for sorting and hence can be applied for selection as well. On the other hand, the algorithm given in [63] is very simple. [63]'s algorithm has been implemented on CM-2 and empirical results are promising [70]. The best known deterministic algorithm is due to Berthomé et. al. [13] and has a run time of $O(\log n \log^* n)$.

For the case of $p < n$ on the sequential model, [61]'s deterministic algorithm runs in time $O(\frac{n}{p}\log\log p + \log^2 p \log(\frac{n}{p}))$ whereas the randomized algorithm of [63] has a run time of $\widetilde{O}(\frac{n}{p}\log\log p + \log p \log\log p)$. A lower bound for this problem is $\frac{n}{p}\log\log p + \log p$. On the weak parallel model, [61]'s deterministic algorithm has a run time of $O(\frac{n}{p} + \log p \log\log p)$ and the randomized algorithm of [63] has a run time of $\widetilde{O}(\frac{n}{p} + \log p)$. A lower bound for selection on this model is $\frac{n}{p} + \log p$. All of these algorithms 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 [70]. The run time is $O(\frac{n}{p}\log\log p + \log^2 p \log\log p)$. If a better sorting algorithm is discovered for the hypercube, this time bound will improve further.

4.6 THE STAR GRAPH

The only known selection algorithm on the star graph is due to Rajasekaran and Wei [76]. This randomized algorithm runs in time $\widetilde{O}(n^2)$ 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 $\widetilde{O}(n^3)$ follows from this algorithm and is discussed in section 3.7.

4.7 PARALLEL DISK SYSTEMS

In [68] two algorithms have been presented for selection on the PDS model. The first algorithm is randomized and the second algorithm is deterministic. The number of parallel I/O read operations needed for either is $O\left(\frac{N}{DB}\right)$, where N is the number of input keys, D is the number of disks, and B is the block size.

Thus the algorithms are asymptotically optimal. Due to the small underlying constants, the algorithms have the potential of being practical as well. The randomized algorithm is based the general theme given above.

5. CONCLUSIONS

In this article we have surveyed known parallel algorithms for sorting and selection on various models of computing. We have also identified some very commonly used techniques for the design of such algorithms.

References

[1] A. Aggarwal and C. G. Plaxton, Optimal Parallel Sorting in Multi-Level Storage, *Proc. Fifth Annual ACM Symposium on Discrete Algorithms*, 1994, pp. 659-668.

[2] A. Aggarwal and J. S. Vitter, The Input/Output Complexity of Sorting and Related Problems, *Communications of the ACM* 31(9), 1988, pp. 1116-1127.

[3] A.V. Aho, J.E. Hopcroft, and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley Publishing Company, 1974.

[4] 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.

[5] M. Ajtai, J. Komlós, W. Steiger, and E. Szemerédi, Optimal Parallel Selection Has Complexity $O(\log \log n)$, *Journal of Computer and System Science*, 38, 1989, pp. 125-133.

[6] N. Alon and Y. Azar, The Average Complexity of Deterministic and Randomized Parallel Comparison Sorting Algorithms, *SIAM Journal of Computing*, 17, 1988, pp. 1178-1192.

[7] N. Alon, Y. Azar and U. Vishkin, Tight Complexity Bounds for Parallel Comparison Sorting, *Proc. of 29th IEEE Symposium on Foundations o Computer Science*, 1986, pp. 502-510.

[8] Y. Azar and N. Pippenger, Parallel Selection, *Discrete Applied Mathematics*, 27, 1990, pp. 49-58.

[9] R. Barve, E. F. Grove, and J. S. Vitter, Simple Randomized Mergesort on Parallel Disks, *Parallel Computing* 23(4-5), 1997, pp. 601-631.

[10] K.E. Batcher, Sorting Networks and their Applications, *Proc. 1968 Spring Joint Computer Conference*, vol. 32, AFIPS Press, 1968, pp. 307-314.

[11] 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.

[12] Y. Ben-Asher, D. Peleg, R. Ramaswami, and A. Schuster, The Power of Reconfiguration, *Journal of Parallel and Distributed Computing*, 1991, pp. 139-153.

[13] 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.

[14] G.E. Blelloch, C.E. Leiserson, B.M. Maggs, C.G. Plaxton, S.J. Smith, M. Zagha, A Comparison of Sorting Algorithms for the Connection Machine CM-2, *Proc. 3rd Annual ACM Symposium on Parallel Algorithms and Architectures*, 1991, pp. 3-16.

[15] 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.

[16] R. Boppana, The Average-Case Parallel Complexity of Sorting, *Information Processing Letter*, 33, 1989, pp. 145-146.

[17] J.C. Cogolludo and S. Rajasekaran, Permutation Routing on Reconfigurable Meshes, *Proc. Fourth International Symposium on Algorithms and Computation*, Springer-Verlag Lecture Notes in Computer Science 762, 1993, pp. 157-166.

[18] R. Cole, Parallel Merge Sort, *SIAM Journal on Computing*, vol. 17, no. 4, 1988, pp. 770-785.

[19] R. Cole, An Optimally Efficient Selection Algorithm, *Information Processing Letters*, 26, 1988, pp. 295-299.

[20] A. Condon and L. Narayanan, Upper and Lower Bounds for Selection on the Mesh, *Algorithmica*, 30, 1998, pp. 1-30.

[21] 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.

[22] 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.

[23] H. ElGindy and P. Wegrowicz, Selection on the Reconfigurable Mesh, *Proc. International Conference on Parallel Processing*, 1991, Vol. III, pp. 26-33.

[24] R.W. Floyd and R.L. Rivest, Expected Time Bounds for Selection, *Communications of the ACM*, 18(3), 1975, pp. 165-172.

[25] 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.

[26] T. Hagerup and R. Raman, An Optimal Parallel Algorithm for Selection, *Proc. 5th Annual ACM Symposium on Parallel Algorithms and Architectures*, 1993, pp. 346-355.

[27] E. Hao, P.D. McKenzie and Q.F. Stout, Selection on the Reconfigurable Mesh, *Proc. Frontiers of Massively Parallel Computation*, 1992, pp. 38-45.

[28] 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.

[29] C.A.R. Hoare, Quicksort, *The Computer Journal*, 5, 1962, pp. 10-15.

[30] E. Horowitz, S. Sahni, and S. Rajasekaran, *Computer Algorithms*, W.H. Freeman Press, 1998.

[31] D.F. Hsu, D.S.L. Wei, Permutation Routing and Sorting on Directed de Bruijn Networks, Technical Report, University of Aizu, Japan, 1994.

[32] 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.

[33] J. Jang and V.K. Prasanna, An Optimal Sorting Algorithm on Reconfigurable Mesh, *Proc. International Parallel Processing Symposium*, 1992, pp. 130-137.

[34] 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.

[35] 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, pp. 17-28.

[36] 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.

[37] D. Krizanc, Time-Randomness Tradeoffs in Parallel Computation, *Journal of Algorithms*, 20, 1996, pp. 1-19.

[38] D. Krizanc, and L. Narayanan, Optimal Algorithms for Selection on a Mesh-Connected Processor Array, *Proc. IEEE Symposium on Parallel and Distributed Processing*, 1992, pp. 70-76.

[39] D. Krizanc and L. Narayanan, Multi-packet Selection on a Mesh-Connected Processor Array, *Proc. International Parallel Processing Symposium*, 1992, pp. 602-605.

[40] D. Krizanc, L. Narayanan and R. Raman, Fast Deterministic Selection on a Mesh-Connected Processor Array, *Algorithmica*, 15, 1996, pp. 319-332.

[41] D. Krizanc, S. Rajasekaran, and S. Shende, A Comparison of Meshes with Static Buses and Unidirectional Wrap-Arounds, *Parallel Processing Letters*, 3, 1993, pp. 119-114.

[42] V.K.P. Kumar and C.S. Raghavendra, Array Processor with Multiple Broadcasting, *Journal of Parallel and Distributed Computing*, 4, 1987, pp. 173-190.

[43] 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.

[44] 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.

[45] T. Leighton, Tight Bounds on the Complexity of Parallel Sorting, *IEEE Transactions on Computers*, C34(4), 1985, pp. 344-354.

[46] T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays–Trees–Hypercube*, Morgan-Kaufmann Publishers, 1992.

[47] H. Li and Q. Stout, *Reconfigurable Massively Parallel Computers*, Prentice-Hall Publishers, 1991.

[48] 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, pp. 16-27.

[49] 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.

[50] N. Meggido, Parallel Algorithms for Finding the Maximum and the Median Almost Surely in Constant Time, Technical Report, School of Computer Science, Carnegie-Mellon University, Pittsburg, PA, Oct. 1982.

[51] 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.

[52] 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.

[53] J.I. Munro and M.S. Paterson, Selection and Sorting with Limited Storage, *Theoretical Computer Science*, 12, 1980, pp. 315-323.

[54] K. Nakano, D. Peleg, and A. Schuster, Constant-time Sorting on a Reconfigurable Mesh, Manuscript, 1992.

[55] 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.

[56] M. Nigam and S. Sahni, Sorting n^2 Numbers on $n \times n$ Meshes, *Proc. International Parallel Processing Symposium*, 1993, pp. 73-78.

[57] 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.

[58] M. H. Nodine and J. S. Vitter, Large Scale Sorting in Parallel Memories, *Proc. Third Annual ACM Symposium on Parallel Algorithms and Architectures*, 1990, pp. 29-39.

[59] V. S. Pai, A. A. Schaffer, and P. J. Varman, Markov Analysis of Multiple-Disk Prefetching Strategies for External Merging, *Theoretical Computer Science*, 128(2), 1994, pp. 211-239.

[60] M. D. Pearson, Fast Out-of-Core Sorting on Parallel Disk Systems, Technical Report PCS-TR99-351, Dartmouth College, Computer Science, Hanover, NH, June 1999, `ftp://ftp.cs.dartmouth.edu/TR/TR99-351.ps.Z`.

[61] C.G. Plaxton, Efficient Computation on Sparse Interconnection Networks, Ph. D. Thesis, Department of Computer Science, Stanford University, 1989.

[62] F. Preparata, New Parallel Sorting Schemes, *IEEE Transactions on Computers*, C27(7), 1978, pp. 669–673.

[63] S. Rajasekaran, Randomized Parallel Selection, *Proc. Symposium on Foundations of Software Technology and Theoretical Computer Science*, 1990, pp. 215-224.

[64] 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.

[65] S. Rajasekaran, $k-k$ Routing, $k-k$ Sorting, and Cut Through Routing on the Mesh, *Journal of Algorithms* 19, 1995, pp. 361-382.

[66] S. Rajasekaran, Mesh Connected Computers with Fixed and Reconfigurable Buses: Packet Routing and Sorting, *IEEE Transactions on Computers*, 45(5), 1996, pp. 529-539.

[67] S. Rajasekaran, A Framework For Simple Sorting Algorithms On Parallel Disk Systems, *Proc. 10th Annual ACM Symposium on Parallel Algorithms and Architectures*, 1998, pp. 88-97.

[68] S. Rajasekaran, Selection Algorithms for the Parallel Disk Systems, *Proc. International Conference on High Performance Computing*, 1998, pp. 103-110.

[69] S. Rajasekaran, Selection on Mesh Connected Computers with Fixed and Reconfigurable Buses, *Journal of Algorithms*, 29, 1998, pp. 68-81.

[70] S. Rajasekaran, W. Chen, and S. Yooseph, Unifying Themes for Parallel Selection, *Proc. Fourth International Symposium on Algorithms and Computation*, Springer-Verlag Lecture Notes in Computer Science 834, 1994, pp. 92-100.

[71] S. Rajasekaran and X. Jin, A Practical Realization of Parallel Disks, to appear in *Proc. International Workshop on High Performance Scientific and Engineering Computing with Applications*, 2000.

[72] 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.

[73] S. Rajasekaran and J.H. Reif, Optimal and Sub-Logarithmic Time Randomized Parallel Sorting Algorithms, *SIAM Journal on Computing*, 18(3), 1989, pp. 594-607.

[74] 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.

[75] 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.

[76] S. Rajasekaran and D.S.L. Wei, Selection, Routing, and Sorting on the Star Graph, *Proc. International Parallel Processing Symposium*, 1993, pp. 661-665.

[77] J.H. Reif and L.G. Valiant, A Logarithmic Time Sort for Linear Size Networks, *Journal of the ACM*, 34(1), 1987, pp. 60-76.

[78] R. Reischuk, Probabilistic Parallel Algorithms for Sorting and Selection, *SIAM Journal of Computing*, 14(2), 1985, pp. 396-409.

[79] C. Schnorr and A. Shamir, An Optimal Sorting Algorithm for Mesh-Connected Computers, *Proc. ACM Symposium on Theory of Computing*, 1986, pp. 255-263.

[80] J.F. Sibeyn, M. Kaufmann, and R. Raman, Randomized Routing on Meshes with Buses, *Proc. European Symposium on Algorithms*, 1993, pp. 333-344.

[81] 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.

[82] 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*, 1988, pp. 437-442.

[83] C.D. Thompson and H.T. Kung, Sorting on a Mesh Connected Parallel Computer, *Communications of the ACM*, 20(4), 1977, pp. 263-271.

[84] L. G. Valiant, Parallelism in Comparison Problems, *SIAM Journal of Computing*, 4, 1975, pp. 348-355.

[85] J. S. Vitter and E. A. M. Shriver, Algorithms for Parallel Memory I: Two-Level Memories, *Algorithmica* 12(2-3), 1994, pp. 110-147.

[86] 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), 1990, pp. 187-192.