Efficient Parallel Hierarchical Clustering Algorithms

Sanguthevar Rajasekaran, *Senior Member*, *IEEE*

Abstract—Clustering of data has numerous applications and has been studied extensively. Though most of the algorithms in the literature are sequential, many parallel algorithms have also been designed. In this paper, we present parallel algorithms with better performance than known algorithms. We consider algorithms that work well in the worst case as well as algorithms with good expected performance.

Index Terms—Reconfigurable networks, mesh-connected computers, meshes with optical buses, hierarchical clustering, PRAMs, single-link metric.

1 Introduction

Clustering of data in multidimensions has many applications. The idea of clustering is to partition the data into groups such that each group has "similar" data items. Many methods are known for clustering. Two popular techniques are hierarchical clustering (see, e.g., [12], [23]) and squared error clustering (see, e.g., [9], [20]).

The basic principle behind hierarchical clustering is the following: If there are n input points (or data items), we start with n clusters where each cluster has a single point. From there on, the "closest" two clusters are identified. The distance between two clusters can be defined in many ways. The two closest clusters are merged, resulting in a reduction in the number of clusters (by one). This process of merging is continued until the number of remaining clusters is q (where q is the target number of clusters and could be a part of the input).

We focus in this paper on parallel algorithms for hierarchical clustering. There are many ways in which the distance between clusters can be defined [12]. The metric that is commonly employed is the *single link* metric. Here, the distance between two clusters is defined to be the minimum weight of any edge connecting a point in one cluster with a point in the other. In this paper, we employ the single link metric as well.

Several papers have been written on the topic of parallel hierarchical clustering (under the single link metric) on various models of computing. Sequential algorithms with a runtime of $O(n^2)$ are known [12]. Olson has presented $O(n \log n)$-time $\frac{n}{\log n}$-processor algorithms on the Parallel Random Access Machine (PRAM), butterfly, and tree models. In this paper, we present a PRAM algorithm that runs in $O(\log n)$-time using $\frac{n^2}{\log n}$ Concurrent-Read-Concurrent-Write (CRCW) PRAM processors. Li and Fang's algorithm [10] runs in $O(n \log n)$-time using n SIMD hypercube processors. An $O(n^2)$-time n-processor SIMD shuffle-exchange network algorithm has been given by Li [8]. Tsai et al. [22] have presented an $O(\log^2 n)$-time algorithm that uses n^3-processor array with a reconfigurable bus system (PARBS) processors. Wu et al. [23] have presented an $O(\log n)$-cycles algorithm that uses n^3 processors on the Arrays with Reconfigurable Optical Buses (AROB) model. They also present $O(1)$-cycles algorithms for the AROB. In this paper, we present an AROB algorithm that runs in $O(\log^2 n)$-cycles using n^2 processors.

In this paper, we also present parallel algorithms that perform well on the average. We present a PRAM algorithm that runs in an expected $O(\log n)$-time using n processors (assuming that the dimension is a constant). The same techniques run in an expected $O(\log^2 n)$ cycles on a $1 \times n$ AROB. These algorithms are perhaps of only theoretical interest since the assumption of uniform distribution of points made in these algorithms may not hold in practice.

2 Preliminaries

2.1 Models of Computing

The PRAM is a collection of Random Access Machines (RAMs) wherein interprocessor communications take place by writing into and reading from common memory cells. Different versions of the PRAM have been identified depending on how read and write conflicts are resolved. (See, e.g., [5], [4].) In an Exclusive Read Exclusive Write (EREW) PRAM, neither concurrent reads nor concurrent writes are permitted. Concurrent Read Exclusive Write (CREW) PRAM permits concurrent reads, but prohibits concurrent writes. CRCW PRAM permits both concurrent reads and concurrent writes. Many variations of a CRCW PRAM are possible based on how write conflicts are handled. In a common CRCW PRAM, concurrent writes are permitted if and only if all the processors attempting a concurrent write have the same message to write. In an arbitrary CRCW PRAM, in cases of attempts on concurrent writes, an arbitrary processor succeeds in writing. In a

• *The author is with the Department of Computer and Science Engineering, University of Connecticut, Storrs, CT 06269.*
E-mail: rajasek@engr.uconn.edu.

Manuscript received 31 Dec. 2003; revised 19 July 2004; accepted 4 Nov. 2004; published online 21 Apr. 2005.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0246-1203.

priority CRCW PRAM, write conflicts are resolved using priorities assigned to processors.

An AROB is nothing but a k-dimensional (for $k \geq 1$) reconfigurable mesh [6] in which the buses are implemented using optical technology. In a reconfigurable mesh, the switch in each processor can be used to connect together subsets of the four bus segments connected to the processor. Reconfigurable meshes that use electronic buses have been studied extensively. Reconfigurable meshes with optical buses have been less extensively studied. An AROB differs from a reconfigurable mesh in several ways. We briefly summarize the features of an AROB below. More details on the model can be found in [13], [11], [23].

In the AROB model of [13], [11], the allowable switch settings of the processors are the same as those in the RN model of [1]. A bus link connects two adjacent processors x and y and has two associated wave guides. One of the wave guides permits an optical signal to travel from x to y and the other permits signal movement from y to x. By setting processor switches, bus links are connected together to form disjoint buses. On each bus, we need to specify which orientation of the waveguide on each link of the bus is to be used. The resulting directed graph that represents the bus should be a directed chain. The root of this chain is the bus "leader." The length of a bus is the number of links on the chain representing that bus. The position of any processor on a bus is its distance from the bus leader. The time needed to transmit a message on a bus is referred to as one cycle. A cycle is divided into slots of duration τ and each slot can carry a different optical signal. τ is the time needed for an optical pulse to move down one bus link. In particular, τ encompasses the time to send a b-bit message where each bit is a light pulse with a w second duration (for appropriate values of $b \geq 1$ and w). The runtimes of AROB algorithms are specified in terms of cycles. Pavel and Akl [13] have argued that the duration of a cycle is comparable to the time for a CPU operation. Even though bus cycles are fast, the assumption that a cycle corresponds to $O(1)$ time may not hold for arbitrarily large bus lengths. Numerous papers have been written based on this model. For more details on the model, the reader is referred to [13] and [23]. In fact, the AROB version assumed in this paper is weaker than the ones employed in [23].

If a parallel algorithm that uses P processors runs in time T, then the *work done* by this parallel algorithm is defined to be PT.

2.2 Preliminary Results for the AROB

We consider some known AROB algorithms in this section. These algorithms are employed in our hierarchical clustering algorithms.

Lemma 2.1. *Consider a $1 \times n$ processor AROB. If each processor has a bit, then the prefix sums of these bits can be computed in $O(1)$ cycles [13].*

The idea behind the above algorithm is as follows: Processor 1 initiates a light pulse in time slot one of a cycle if its bit is zero and in slot two otherwise. All processors start their counters at the start of the cycle and also set their delay units to introduce a one slot delay in case the processor's bit is one. A processor's counter is turned off when the light pulse initiated by processor 1 reaches it. By using the terminal counter value, its data bit, and its distance from processor 1, each processor can compute its prefix sum value.

In any fixed connection network, a single step of interprocessor communication can be thought of as a packet routing task. The problem of routing can be stated as follows: There is a packet of information at each node that is destined for some other node. Send all the packets to their correct destinations as quickly as possible, making sure that at most one packet crosses any edge at any time. The *runtime* of any packet routing algorithm is defined to be the time taken by the last packet to reach its destination. The *queue size* is the maximum number of packets that any processor will have to store during the algorithm.

The problem of *partial permutation routing* is the task of routing where at most one packet originates from any node and at most one packet is destined for any node. Any routing problem where at most h packets originate from any node and at most h packets are destined for any node will be called $h-h$ *routing* or h-relations *routing*.

Lemma 2.2. *In an AROB of size $1 \times n$, any permutation can be routed in $O(1)$ cycles [13].*

The above lemma has been strengthened as follows in [19].

Lemma 2.3. *Let $\mathcal{L}$ be an AROB of size $1 \times n$. Consider a routing problem where $O(1)$ packets originate from any node and $O(1)$ packets are destined for any node. This problem can also be solved in $O(1)$ cycles.*

An extension of the above lemmas yields the following lemma [13].

Lemma 2.4. *Say there are k elements arbitrarily distributed (at most one per processor) in a two-dimensional AROB of size $\sqrt{n} \times \sqrt{n}$. We would like to "compact" them in the first $\lceil \frac{k}{\sqrt{n}} \rceil$ rows. This problem can be solved in $O(1)$ cycles.*

The following lemma is proven in [14]:

Lemma 2.5. *Given n integers of bounded magnitude, these numbers can be sorted in $O(1)$ cycles on a $1 \times n$ AROB.*

2.3 Hierachical Clustering Primitives

In hierarchical clustering, we start with single point clusters and merge clusters in stages until the desired number of clusters results. Alternatively, one could create and store a hierachical cluster structure (known as the *dendrogram*) corresponding to $q = 1$ [12]. The dendrogram is nothing but a tree that shows which clusters were merged at each step. In fact, this tree can be easily constructed from a *Euclidean minimum spanning tree* of the n input points.

The Euclidean minimum spanning tree problem (EMSTP) is defined as follows: Input are n points in a Euclidean space. Consider a graph $G(V, E)$ whose nodes are the points in this space. There is an edge between every pair of nodes in V. The weight on any such edge is the Euclidean distance between those two points. The problem is to find a minimum spanning tree for this graph. Efficient

algorithms for finding EMSTs are crucial for solving the clustering problem under the single-link metric.

The dendrogram can be used to obtain clusters of interest. For instance, we may be supplied with a threshold distance d. We may be required to output clusters corresponding to this threshold distance. The corresponding clusters can be obtained as follows: We remove all the edges in the dendrogram whose weights (or distances) are greater than d. The connected components in the resultant forest are the clusters of interest.

On the other hand, let $T(V, E')$ be a Euclidean minimum spanning tree for the input points and let q be the number of clusters that we are interested in. We could use the dendrogram to find these clusters. Alternatively, we could use T as well to identify these clusters as follows: Let E'' be the set of $|V| - q$ edges in E' with minimum weights. The connected components of $T(V, E'')$ are the clusters of interest. In this paper, we suggest the use of this technique of identifying clusters. In other words, we compute the EMST for the input points and keep it. We do not build the dendrogram. As and when a q is input, we compute the clusters from T.

In summary, finding EMSTs of n given points and identifying the connected components of a graph are the two basic problems underlying the hierarchical clustering problem under the single link metric. Thus, papers concentrating on solving the hierarchical clustering problem focus mainly on solving these two problems (see, e.g., [12], [23]).

2.4 Euclidean Minimum Spanning Trees

The problem of finding a minimum spanning tree of a given weighted graph $G(V, E)$ has been studied extensively and numerous (sequential, randomized, and parallel) algorithms have been devised for its solution. Asymptotically optimal randomized algorithms have been developed (see, e.g., [7]). These algorithms run in time $O(|V| + |E|)$ with high probability. Here, the high probability is over the space of all possible outcomes for the coin flips made in the algorithm. A deterministic algorithm with a runtime of $O(|E|\alpha(|E|, |V|))$ has been given in [2] where α is the inverse Ackerman's function.

Clearly, the algorithms mentioned above could be employed to solve the EMST problem. In this case, the runtime will be $O(n^2)$ (or nearly $O(n^2)$).

Shamos and Hoey [21] have presented an $O(n \log n)$ time algorithm for this problem, assuming that the points are from a two-dimensional space. Yao [24] considers the EMSTP in k-dimensions (for $k \geq 3$). In particular, he has presented an algorithm with a runtime of $O(n^{2-a(k)}(\log n)^{1-a(k)})$, where $a(k) = 2^{-(k+1)}$. When $k = 3$, an improved algorithm with a runtime of $O((n \log n)^{1.8})$ has also been given in [24].

Recently, it has been pointed out that the EMST problem can be solved in k-dimensions in an expected $O(n)$ time, as long as k is a constant [17]. Here, the expectation is under the assumption that the input points are uniformly distributed in the space.

In this section, we adapt the algorithm of [17] to develop a parallel algorithm which runs in an expected $O(\log n)$-time using $\frac{n}{\log n}$ PRAM processors.

The idea behind the sequential algorithm of [17] is the following: A graph $G(V, E)$ is generated where V is the set of input points and E consists of edges from every input point p to points within a certain neighborhood of p. $|E| = O(|V|)$ with high probability. By high probability, we mean a probability of $\geq (1 - n^{-\alpha})$ for any fixed $\alpha \geq 1$, where $n = |V|$. We use the algorithm of [7] (for example) to find a minimum spanning forest F of $G(V, E)$. If F has only one tree, then we output this and quit. Let $T(V', E')$ be the tree in F that has the largest number of nodes. Let $V'' = V - V'$. For every point p in V'' we include edges from p to points in a sufficiently large neighborhood of p. If these edges, together with E', are enough to produce a minimum spanning tree for all the input points, we output this and quit. If not we expand the neighborhood of points in V'' and repeat this process until we succeed.

Let the input points be $p_1, p_2, \ldots, p_n$. Without loss of generality, assume that these points are uniformly distributed within a unit square in the plane. In particular, if (x, y) is an input point, then both x and y are independent random variables uniformly distributed in the interval $(0, 1)$.

Partition the square into a $\sqrt{n} \times \sqrt{n}$ square grid with n cells, the size of each cell being $\frac{1}{\sqrt{n}} \times \frac{1}{\sqrt{n}}$. We refer to the length $\frac{1}{\sqrt{n}}$ as one *unit* from hereon. Label these cells $g_{i,j}$ for $1 \leq i, j \leq \sqrt{n}$. The cell $g_{i,j}$ is in the ith row and jth column. The expected number of points in each cell is 1. For any input point p, define its x-neighborhood to be a circle of radius x units centered at p. For example, for every input point p, we can find all the input points that are within a C-neighborhood of p in $O(n)$ time (C being a constant). The idea is to:

1. For each point p, find the cell coordinates (i, j) of p (i.e., p is in $g_{i,j}$).
2. Sort the points according to their cell coordinates (using radix sort). For each cell $g_{i,j}$, let $L_{i,j}$ be the list of points in $g_{i,j}$.
3. To identify the points in a C-neighborhood of p, just examine the appropriate ($O(1)$) cells surrounding $g_{i,j}$.

The adjacency list representation of the graph $G(V, E)$ for the n given points can be constructed in parallel as follows. Step 1 above can be done easily within the stated bounds. Sorting in Step 2 can be done in $O(\log n)$ time with high probablity using $\frac{n}{\log n}$ CRCW PRAM processors using the algorithm of [18]. Once the sorting is done, we can form the n adjacency lists easily in an expected $O(n)$ work. Once we construct a graph $G(V, E)$ with an expected $O(n)$ edges, we can use the following randomized algorithm (due to Pettie and Ramachandran [15]) to find an MST of G:

Lemma 2.6. *The minimum spanning tree problem on a weighted graph $G(V, E)$ can be solved in $O(\log |V|)$ time (with high probability) on the EREW PRAM using $O\left(\frac{|V|+|E|}{\log |V|}\right)$ processors.*

The remaining steps of the algorithm of [17] can also be done similarly. Thus, we get the following:

Theorem 2.1. *Given n points in the plane, we can find an EMST in an expected $O(\log n)$-time on the CRCW PRAM. The expected work done is $O(n)$ using a randomized algorithm. This algorithm produces the correct answer for a large fraction of all possible inputs.*

The above algorithm can be extended to higher dimensions as well. Consider a space of dimension k, where k is a constant. Assume, without loss of generality, that we are given n input points in a unit cube. We can partition the cube into subcubes of dimension $\frac{1}{n^{1/k}} \times \frac{1}{n^{1/k}} \times \cdots \times \frac{1}{n^{1/k}}$ each. The expected number of points in any subcube is only $O(1)$.

We can use the same idea as above and get the following theorem:

Theorem 2.2. *We can find an EMST for n given points in parallel in a space of dimension k in $O\left(nC_1^k + n^{\beta}(C_2 \log n)^k\right)$ expected work using a randomized algorithm. Here, β is a constant < 1 (see [17]). The expected runtime on the CRCW PRAM is $O(\log n)$. This algorithm outputs the correct answer for a large fraction of all possible inputs.*

3 Clustering Algorithms with Worst-Case Guarantees

In this section, we present parallel algorithms for hierarchical clustering which run in $O(\log n)$-time and whose worst-case work done is better than those of all the existing algorithms.

3.1 A PRAM Algorithm

As has been pointed out before, hierarchical clustering (under the single-link metric) can be solved with efficient algorithms for the EMSTP and the connected components problem.

A corollary to Lemma 2.6 is stated below.

Corollary 3.1. *The EMST problem on n points can be solved in $O(\log n)$ time using $\frac{n^2}{\log n}$ CRCW PRAM processors.*

Optimal algorithms are known for finding the connected components of graphs, as the following lemma indicates [3]:

Lemma 3.1. *The connected components of any graph $G(V, E)$ can be found in $O(\log |V|)$ time using a randomized algorithm on a CRCW PRAM, the work done being $O(|V| + |E|)$.*

The algorithm of [3] is very complex. For the hierarchical clustering problem, the graph for which connected components are needed is a forest. Moreover, we have $\frac{n^2}{\log n}$ processors. Algorithm 5.2 given in [5] can be employed in this context. A specialization of this algorithm to our case yields the following lemma:

Lemma 3.2. *The connected components of a forest $T(V, E)$ can be found in $O(\log |V|)$ time on a CRCW PRAM, the total work being $O(|V| \log |V|)$.*

Corollary 3.1 and Lemma 3.2 together yield the following theorem:

Theorem 3.1. *The hierarchical clustering problem can be solved in $O(\log n)$ time on the CRCW PRAM, the total work done being $O(n^2)$.*

3.2 An AROB Algorithm

In this section, we present an algorithm for the AROB model. This algorithm runs in $O(\log^2 n)$ cycles using n^2 processors. In contrast, the algorithm of [23] uses n^3 processors and runs in $O(\log n)$ cycles. The algorithm of [22] runs in $O(\log^2 n)$ cycles using an n^3 processor array with reconfigurable bus system (PARBS) processors.

Connected Components. First, we consider the connected components problem. The input is a forest $T(V, E)$ on n nodes. Specifically, the input consists of a list of q edges. These edges form $n - q$ trees. We are required to label the nodes such that nodes in the same tree get the same label. Let $V = \{1, 2, \ldots, n\}$.

In a $1 \times n$ AROB, assume that the edges are input one edge per processor. At the end of the algorithm, the label of node i will be in the ith processor. If we wish to output the connected components (as $n - q$ lists, one corresponding to each component), this can be done in an additional $O(1)$ cycles by sorting the labels of nodes (see Lemma 2.5).

The idea is to use Algorithm 5.2 given in [5]. This algorithm finds the connected components of a graph $G(V, E)$ in $O(\log n)$-time (where $n = |V|$) on a CRCW PRAM, as stated in the following theorem:

Theorem 3.2. *The connected components of a given undirected graph $G(V, E)$ can be found in $O(\log |V|)$-time on a CRCW PRAM, the total work done being $O((|V| + |E|) \log |V|)$.*

In addition to the above theorem, we employ the following lemma as well:

Lemma 3.3. *Each step of an n-processor CRCW PRAM with n common memory cells can be simulated in $O(1)$ cycles on a $1 \times n$ AROB.*

Proof. Assume that common memory cell i is with processor i, $1 \leq i \leq n$. The idea is similar to the one used for permutation routing (c.f. Lemma 2.3) (see, e.g., [19]). The time it takes for a packet to move from one processor to the next is assumed to be τ. Consider any concurrent read step. Let the processors be numbered $1, 2, \ldots, n$, starting from the left. There is a time slot assigned to each processor for reading from (and writing into) the bus. Processor 1 creates a "time slot" for each packet that moves one edge per τ time. The time slots created will be in the order of the processors, i.e., the first time slot is meant for processor 1, time slot 2 is meant for processor 2, and so on. Processor p writes the contents of memory cell p at time $t + p\tau$ (for $1 \leq p \leq n$), where t is the start time. In other words, processor p writes in slot p. If processor q (for $q > p$) wants to read memory cell p, it reads at time $t + (p + q)\tau$, where t is the start time. For example, processor q reads from slot p. Clearly, this algorithm terminates in two cycles (or $2n\tau$ time). The case of $q < p$ is handled by the second wave guide. An arbitrary CRCW PRAM write step can also be handled in exactly the same manner. □

Theorem 3.2 and Lemma 3.3 yield the following lemma:

Lemma 3.4. *We can solve the connected components labeling problem in $O(\log n)$ cycles on a $1 \times n$ AROB, when the input is a forest on n nodes.*

EMSTP. We now concentrate on the EMST problem. We adapt Sollin's algorithm (see, e.g., [5]). This algorithm is based on the following lemma [5]:

Lemma 3.5. *Consider a weighted graph $G(V, E)$. Let $V_1, V_2, \ldots, V_m$ be an arbitrary partition of V. Let e_i be the minimum weight edge connecting a vertex in V_i to a vertex in $V - V_i$, for $1 \leq i \leq m$. Then, the m edges $e_1, e_2, \ldots, e_m$ all belong to an MST of $G(V, E)$.*

There are $O(\log n)$ phases in Sollin's algorithm. The algorithm has a forest of trees at any time. When the number of trees in the forest is one, the algorithm terminates. To begin with, the forest has $n(= |V|)$ trees, one corresponding to each node in the graph. Let F_s denote the forest at the beginning of phase s ($s = 1, 2, \ldots$). Phase s of the algorithm runs as follows: For each tree $T \in F_s$, determine the edge e_T with the minimum weight that connects a node in T to a node outside T. Add these edges to the forest F_s to obtain F_{s+1}. Clearly, the number of trees in F_{s+1} is at most one half of the number of trees in F_s. Thus, the algorithm terminates in $O(\log n)$ phases.

Now, we demonstrate that each phase of Sollin's algorithm can be implemented on an $n \times n$ AROB in $O(\log n)$ cycles. Label the processors as $P_{i,j}, 1 \leq i, j \leq n$. $P_{i,j}$ denotes the processor in the ith row and the jth column of the AROB.

We assume that the input points are given in the first row of the AROB. Let the points be $p_1, p_2, \ldots, p_n$. In any phase of the algorithm, each tree in the forest is represented as a single supernode. If i and j are two such supernodes, then only one edge between these supernodes is kept. This edge is nothing but the edge whose weight is minimum and which connects a node in the tree of i to a node in the tree of j. Also, if there are m trees in any phase of the algorithm, an $m \times m$ matrix is used to denote the edge weights. This matrix is stored in the AROB in the top left subarray of size $m \times m$ (i.e., in the first m rows and the first m columns of the AROB).

To begin with, the weight matrix is computed as follows. Broadcast p_j along column j for $1 \leq j \leq n$ in parallel. At the end each row gets a copy of the input points. Followed by this, p_i is broadcast along row i (for $1 \leq i \leq n$ in parallel). After this, processor $P_{i,j}$ computes the distance between p_i and p_j (for $1 \leq i, j \leq n$ in parallel).

Algorithm AROB_Clustering
$m := 1$; $n_m := n$;
while $n_m > 1$ **do**
1) Find the minimum weight edge in each row. Let e_i be this edge from row i, $1 \leq i \leq n_m$.
2) Collect the n_m edges of step 1) in the first row of the AROB (c.f. Lemma 2.4).
3) Find the connected components of the forest defined by the n_m edges. Now, each node knows its supernode number in the new forest. Using a prefix computation, compute the number of trees n_{m+1} in the new forest.
4) Broadcast the new label of node i along row i (for $1 \leq i \leq n_m$ in parallel). Broadcast the new label of node j along column j (for $1 \leq j \leq n_m$ in parallel).
5) Form the new $n_{m+1} \times n_{m+1}$ weight matrix as follows. Sort each row according to the new labels of the nodes. Find the minimum (along each row) corresponding to each group of nodes whose new labels are the same. At the end of this step, each node knows the minimum weight edge originating from it and ending in each supernode. Now, sort each column according to the new labels of the nodes. Find the minimum (along columns) corresponding to each group of nodes whose new labels are the same. At the end of this step, each supernode knows the minimum weight edge to every other supernode.
6) Compact the supernodes in the top left $n_{m+1} \times n_{m+1}$ subarray (c.f. Lemma 2.4).
7) $m := m + 1$.

When the above algorithm terminates, we can restore the marked edges to their original names in the same amount of time. We get the following lemma:

Lemma 3.6. *We can solve the EMST problem in $O(\log^2 n)$-cycles on an $n \times n$ AROB.*

Lemmas 3.4 and 3.6 yield the following theorem:

Theorem 3.3. *The hierarchical clustering problem can be solved in $O(\log^2 n)$ cycles on an $n \times n$ AROB.*

4 Clustering Algorithms with Good Average Performance

In this section, we show that the clustering problem can be solved in an expected $O(\log n)$ time the expected total work being $O(n \log n)$ on the PRAM as long as the data points are from a space of constant dimension. We also show that an expected $O(\log^2 n)$ cycles is achievable on a $1 \times n$ AROB. These results may not be of practical interest since the assumption of uniform distribution of points in the space made in these algorithms may not hold.

4.1 A PRAM Algorithm

We can combine Theorem 2.2 and Lemma 3.2 to arrive at the following theorem:

Theorem 4.1. *The hierarchical clustering problem on n given points in a k-dimensional space can be solved in parallel in $O(n \log n)$ expected work as long as $k = O(1)$. The expected runtime is $O(\log n)$. This algorithm outputs the correct answer for a large fraction of all possible inputs. Also, the expected runtime of this algorithm is exponential in k.*

It is possible to improve the expected work done by the above algorithm to nearly $O(n)$ using randomized or deterministic symmetry breaking techniques (see, e.g., [5] for a description of symmetry breaking).

4.2 An AROB Algorithm

An algorithm similar to AROB_Cluster can be devised that exploits the ideas of Section 4.1 as well. In this case, the expected runtime will be $O(\log^2 n)$ cycles on a $1 \times n$ AROB. We first form a graph with $O(n)$ edges, as explained in Section 2.4, and run Sollin's algorithm on these edges.

Here, again, there are $O(\log n)$ phases, each phase taking $O(\log n)$ cycles. In any phase, we sort the edges appropriately. For instance, to begin with, we are required to find the minimum weight edge incident on every node. This can

be done by just sorting the edges according to the starting point of the edges and then finding group minima.

In any general phase, we are required to find the new weight matrix. The steps illustrated in AROB_Cluster can also be done in a similar fashion with appropriate sorting steps. Thus, we get the following theorem:

Theorem 4.2. *The hierarchical clustering problem can be solved in an expected $O(\log^2 n)$ cycles on a $1 \times n$ AROB as long as k is a constant. The expected runtime of this algorithm is exponential in k.*

5 Conclusions

In this paper, we have given algorithms for the hierarchical clustering problem on the PRAM and the AROB models of computing. We have considered algorithms with worst-case guarantees as well as algorithms with good expected performance. These algorithms have better performance measures than existing algorithms.

Acknowledgments

This research has been supported in part by US National Science Foundation Grants CCR9912395 and ITR0326155.

References

[1] Y. Ben-Asher, D. Peleg, R. Ramaswami, and A. Schuster, "The Power of Reconfiguration," *J. Parallel and Distributed Computing,* pp. 139-153, 1991.
[2] B. Chazelle, "A Minimum Spanning Tree Algorithm with Inverse-Ackerman Type Complexity," *J. ACM,* vol. 47, no. 6, pp. 1028-1047, 2000.
[3] H. Gazit, "An Optimal Randomized Parallel Algorithm for Finding Connected Components in a Graph," *SIAM J. Computing,* vol. 20, no. 6, pp. 1046-1067, 1991.
[4] E. Horowitz, S. Sahni, and S. Rajasekaran, *Computer Algorithms.* W.H. Freeman Press, 1998.
[5] J. JáJá, *An Introduction to Parallel Algorithms.* Addison-Wesley Publishing Company, 1992.
[6] J. Jang and V.K. Prasanna, "An Optimal Sorting Algorithm on Reconfigurable Mesh," *Proc. Int'l Parallel Processing Symp.,* pp. 130-137, 1992.
[7] D.R. Karger, P.N. Klein, and R.E. Tarjan, "A Randomized Linear-Time Algorithm to Find Minimum Spanning Trees," *J. ACM,* vol. 42, no. 2, pp. 321-328, 1995.
[8] X. Li, "Parallel Algorithms for Hierarchical Clustering and Clustering Validity," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 12, pp. 1088-1092, 1990.
[9] X. Li and Z. Fang, "Parallel Algorithms for Clustering on Hypercube SIMD Computers," *Proc. 1986 Conf. Computer Vision and Pattern Recognition,* pp. 130-133, 1986.
[10] X. Li and Z. Fang, "Parallel Clustering Algorithms," *Parallel Computing,* vol. 11, pp. 275-290, 1989.
[11] R.G. Melhem, D.M. Chiarulli, and S.P. Levitan, "Space Multiplexing of Waveguides in Optically Interconnected Multiprocessor Systems," *Computer J.,* vol. 32, no. 4, pp. 362-369, 1989.
[12] C.F. Olson, "Parallel Algorithms for Hierarchical Clustering," *Parallel Computing,* vol. 21, pp. 1313-1325, 1995.
[13] S. Pavel and S.G. Akl, "Matrix Operations Using Arrays with Reconfigurable Optical Buses," manuscript, 1995.
[14] S. Pavel and S.G. Akl, "Integer Sorting and Routing in Arrays with Reconfigurable Optical Buses," *Int'l J. Foundations of Computer Science,* vol. 9, pp. 99-120, 1998.
[15] S. Pettie and V. Ramachandran, "A Randomized Time-Work Optimal Parallel Algorithm for Finding a Minimum Spanning Forest," *Proc. Random-Approx Conf. '99,* pp. 233-244, 1999.
[16] R.C. Prim, "Shortest Connection Networks and Some Generalizations," *Bell System Technical J.* , vol. 36, pp. 1389-1401, 1957.
[17] S. Rajasekaran, "On the Euclidean Minimum Spanning Tree Problem," technical report BECAT/CSE, Univ. of Connecticut, 2004.
[18] S. Rajasekran and J.H. Reif, "Optimal and Sub-Logarithmic Time Randomized Parallel Sorting Algorithms," *SIAM J. Computing,* vol. 18, no. 3, pp. 594-607, 1989.
[19] S. Rajasekaran and S. Sahni, "Sorting, Selection, and Routing on the Array with Reconfigurable Optical Buses," *IEEE Trans. Parallel and Distributed Systems,* vol. 8, no. 11, pp. 1123-1131, Nov. 1997.
[20] S. Ranka and S. Sahni, "Clustering on a Hypercube Multicomputer," *IEEE Trans. Parallel and Distributed Systems,* vol. 2, no. 2, pp. 129-136, 1991.
[21] M.I. Shamos and D.J. Hoey, "Closest-Point Problems," *Proc. 16th Ann. IEEE Symp. Foundations of Computer Science,* pp. 151-162, 1975.
[22] H.R. Tsai, S.J. Horng, S.S. Lee, S.S. Tsai, and T.W. Kao, "Parallel Hierarchical Clustering Algorithms on Processor Arrays with a Reconfigurable Bus System," *Pattern Recognition,* vol. 30, pp. 801-815, 1997.
[23] C.-H. Wu, S.-J. Horng, and H.-R. Tsai, "Efficient Parallel Algorithms for Heirarchical Clustering on Arrays with Reconfigurable Optical Buses," *J. Parallel and Distributed Computing,* vol. 60, pp. 1137-1153, 2000.
[24] A. Yao, "On Constructing Minimum Spanning Trees in k-Dimensional Spaces and Related Problems," *SIAM J. Computing,* vol. 11, no. 4, pp. 721-736, 1982.

Sanguthevar Rajasekaran received the ME degree in automation from the Indian Institute of Science (Bangalore) in 1983 and the PhD degree in computer science from Harvard University in 1988. Currently, he is the UTC chair professor of computer science and engineering at the University of Connecticut. Before joining UConn, he has served as a faculty member in the Computer and Information Science and Engineering Department of the University of Florida and in the Computer and Information Science Department of the University of Pennsylvania. From 2000-2002, he was the chief scientist for Arcot Systems. His research interests include parallel algorithms, bioinformatics, data mining, randomized computing, computer simulations, and combinatorial optimization. He has published more than 120 articles in journals and conferences. He has coauthored two texts on algorithms and coedited four books on algorithms and related topics. He is a senior member of the IEEE.