Parallel Hierarchical Clustering on Shared Memory Platforms

William Hendrix*, Md. Mostofa Ali Patwary, Ankit Agrawal, Wei-keng Liao, and Alok Choudhary
Department of Electrical Engineering and Computer Science
Northwestern University
Evanston, IL 60208
*Corresponding author: whendrix@northwestern.edu

***Abstract*—Hierarchical clustering has many advantages over traditional clustering algorithms like k-means, but it suffers from higher computational costs and a less obvious parallel structure. Thus, in order to scale this technique up to larger datasets, we present SHRINK, a novel shared-memory algorithm for single-linkage hierarchical clustering based on merging the solutions from overlapping sub-problems. In our experiments, we find that SHRINK provides a speedup of 18–20 on 36 cores on both real and synthetic datasets of up to 250,000 points. Source code for SHRINK is available for download on our website, http://cucis.ece.northwestern.edu.**

I. Introduction

Hierarchical clustering is a powerful technique that offers several advantages over traditional partitional clustering techniques, including its non-parametric nature and its ability to elucidate the overall structure of a dataset. Hierarchical clustering has been applied in document classification, bioinformatics, and chemoinformatics (e.g., [1], [2], [3]), as well as others. For example, hierarchical clustering might be used in a bioinformatics context to establish a putative phylogenetic tree among a set of species. However, because it effectively evaluates the cluster structure of a dataset at all levels, hierarchical clustering has a higher computational cost than many traditional clustering algorithms, so a scalable parallel technique is needed in order to apply hierarchical clustering to large datasets. Critically, hierarchical clustering is difficult to parallelize effectively due to high data dependence—each level in the hierarchical dendrogram relies on all of the earlier levels to make sense. Consequently, relatively little work has been done on parallel hierarchical clustering. Moreover, many existing parallel algorithms store a distance matrix, limiting the feasible problem size due to memory constraints. In this paper, we describe SHRINK (SHaRed-memory SLINK), a scalable algorithm for single-linkage hierarchical clustering (SHC) that we implemented in OpenMP. The main parallelization strategy for SHRINK is to divide the original dataset into overlapping subsets, calculate the hierarchical dendrogram for each subset using the state-of-the-art SHC algorithm SLINK [4], and reconstruct the dendrogram for the full dataset by combining the solutions to the subsets. Even though SLINK itself is strongly serial, dividing the dataset allows us to solve the problem in parallel, while reaping the benefits of SLINK, namely, its linear memory requirements and overall efficiency. In addition, OpenMP allows us to read in the dataset once and access it from a single location, rather than making copies of the data.

We evaluate our algorithm empirically on real and synthetic datasets with up to 250,000 data points, and we find that it achieves a speedup of 18–20 on 36 cores across a wide range of datasets. Our main contributions for this work are: (1) SHRINK, a novel shared-memory algorithm for single-linkage hierarchical clustering based on the state-of-the-art serial algorithm SLINK, which we provide as open source on our website;[1] (2) a theoretical analysis of SHRINK, including a formal proof of its correctness, an upper bound on the amount of repeated work, and complexity analysis; and (3) an empirical evaluation of the scalability of the computation and memory requirements for SHRINK on real and synthetic datasets.

The paper is organized as follows: Section II covers related work, Section III describes our proposed algorithm, Section IV covers theoretical aspects of our work, Section V presents our empirical results for SHRINK, and Section VI concludes the paper.

II. Related Work

SLINK, originally created by Sibson [4], is the state-of-the-art algorithm for single-linkage hierarchical clustering (SHC), due to its optimal time and space complexity of $O(n^2)$ and $O(n)$, respectively. SLINK achieves this complexity by iteratively adding one data point at a time to the dataset and updating the dendrogram while only calculating the distance from the new point to the existing points once. Because it does not compute the distance matrix explicitly, it is very effective at solving large problem instances.

Two of the most well-cited works on parallel hierarchical clustering present parallel algorithms, but do not report empirical results. The first of these, described by Bentley [5], distributes the computation to various "tree" processors, while Olson [6] presents two parallel algorithms for single-linkage clustering. The first algorithm from Olson is designed for a shared memory architecture and merges clusters by height, while the second is designed for a butterfly architecture and adds a single point at a time to the dendrogram.

An OpenMP algorithm for parallel SHC was presented by Dash et al. [7]. The algorithm, which is based on partitioning the data using a rectangular grid, achieved a speedup of about

978-1-4673-2371-0/12/$31.00 ©2012 IEEE

5.7 on datasets of up to 30k points using 8 cores. However, this technique computes a distance matrix for each cell of the grid, so it may not be applicable for datasets that are not well-distributed.

Chang et al. [8] present a parallel algorithm for hierarchical clustering on GPUs. Their algorithm parallelizes the process of finding the minimum distance between clusters as well as the pairwise distance computations. They report speedups of up to 48 using an Nvidia Tesla C870 GPU card, though this technique relies on computing a distance matrix, and so might be limited by the size of the dataset it can process.

Du and Lin [9] present a parallel hierarchical clustering algorithm for distributed memory architectures. Their algorithm shows a speedup of 25 on 48 processors on a microarray dataset with 7452 genes across 277 conditions, but it relies on calculating a distributed distance matrix and may have difficulty with larger datasets.

As the problem of single-linkage hierarchical clustering is functionally equivalent to constructing the minimum spanning tree (MST) of a graph in which the vertices represent data points and edges the distances between them, we will mention some work on parallel MST construction. While this problem has been explored more thoroughly, many papers only present theoretical results (e.g., [10], [11], [12]) or present algorithms designed for sparse graphs ([13], [14], [15]). However, Dehne and Götz [16] present a parallel MST algorithm for dense graphs based on dividing the edges of the graph, performing Borůvka's algorithm on the partitions, and combining the resulting MSTs. They report a speedup of 3.5 on 4 processors on two graphs with 1k vertices and 400k edges.

Finally, Olman et al [17] present CLUMP, a parallel clustering algorithm using MPI that is related to both the MST and SHC problems. Olman et al report speedups of up to 100 on 150 processors on a dataset of 1.2 million points; however, they claim that this result is very close to the maximum theoretical speedup.

III. Parallel Single-linkage Hierarchical Clustering

In this section, we will discuss our parallel algorithm for calculating the single-linkage hierarchical clustering (SHC) dendrogram. The main strategy for our algorithm is to break the dataset into overlapping subsets so that we can apply the highly efficient SLINK algorithm to compute the dendrogram for each subset independently and then combine the resulting solutions to form the dendrogram for the full dataset. Specifically, we form these subsets by partitioning the data into k sets and assign each pair of these sets to a thread. In this way, we can decompose the problem into smaller, independent problem instances while still ensuring that every distance calculation is being performed by at least one OpenMP thread. We implement this problem decomposition by forming an array of the data point IDs belonging to each subset. Note that if we choose $k = 2$ partitions initially, SHRINK reduces to the serial SLINK algorithm, with some minor modifications. A pseudocode description of SHRINK appears in Algorithm 1.

Algorithm 1: Pseudocode outline for SHRINK

```
1 Partition the original dataset into k subsets of the roughly
  same size, D_1, D_2, ..., D_k.
2 forall the pairs of subsets (D_i, D_j) do
3     Compute the dendrogram of the dataset D_i ∪ D_j.
      (Algorithm 2)
4 end
5 repeat
6     Combine the dendrograms two at a time by
      processing the cluster merges in order of increasing
      height, eliminating merges that combine data points
      already in the same cluster. (Algorithm 3)
7 until all dendrograms have been combined
8 return the combined dendrogram
```

In step 3 of Algorithm 1, we use a modified version of the SLINK algorithm to calculate a dendrogram for each subset of the original data. This algorithm, which appears in Algorithm 2, differs from SLINK in that:

1) We keep track of the data points used to calculate each distance and store these in the result (Λ) array,
2) We compare data point IDs when distances are equal, and
3) We represent the dendrogram as a sequence of merges between the two closest points in different clusters.

Changes 1 and 2 ensure that the distances between every pair of points in the dataset are effectively distinct, and changes 1 and 3 make it simple to merge the dendrograms. Our modified SLINK implementation is based on the serial `pslcluster` function from the open source Cluster 3.0 library [18].

In steps 6 and 7 of the SHRINK algorithm (Algorithm 1), we combine together all of the dendrograms for the subsets into a single dendrogram that represents the solution of the original problem. These dendrograms are combined by iterating through the cluster merges in order of increasing height, retaining those merges that join data points in different clusters. A full proof of why this procedure generates the dendrogram for the original dataset appears in Section IV-A. To give some intuition on why it works, though, SHC has a strong connection to the minimum spanning tree (MST) problem, and we are essentially combining partial MSTs using Kruskal's algorithm since all of the edges of the MST for the full graph must exist as edges in the MSTs of the subgraphs. Since we sort the merges of the dendrograms by increasing height (see Algorithm 2), we can use a Union-Find (or Disjoint Set) data structure to maintain the cluster membership for each data point as we combine the dendrograms to combine them efficiently [19]. A pseudocode description of the merging procedure appears in Algorithm 3.

Note that steps 6 and 7 in Algorithm 1 do not require that the partial solutions be combined in a particular order. Thus, we can combine them in a binary fashion, using half of the available threads to combine pairs of dendrograms independently in the first step, half as many threads in the

Algorithm 2: Pseudocode description of modified SLINK algorithm [4]. In lines 9, 10, 14, and 19, the height of the merges are compared first, and merges with equal height are compared according to the IDs of the data points they join.

Input: D—Set of data points
Input: d—function to calculate distance between two points
Output: Sorted list of cluster merges in the dendrogram defined by applying a distance function d to every pair of points in D

1 **Algorithm:** MOD-SLINK
2 Initialize each $\Pi[i]$ to be i
3 Initialize the height of each merge in Λ to be ∞
4 **for** $i \leftarrow 1$ to $|D| - 1$ **do**
5 **for** $j \leftarrow 0$ to $i - 1$ **do**
6 M$[j] \leftarrow (D[j], D[i], d(D[j], D[i]))$, representing a merge between $D[j]$ and $D[i]$ at height $d(D[j], D[i])$
7 **end**
8 **for** $j \leftarrow 0$ to $i - 1$ **do**
9 **if** $\Lambda[j]$ has a higher height/id than M$[j]$ **then**
10 **if** $\Lambda[j]$ has a lower height/id than M$[\Pi[j]]$ **then**
11 M$[\Pi[j]] \leftarrow \Lambda[j]$
12 $\Lambda[j] \leftarrow$ M$[j]$
13 $\Pi[j] \leftarrow i$
14 **else if** M$[j]$ has a lower height/id than M$[\Pi[j]]$ **then**
15 M$[\Pi[j]] \leftarrow$ M$[j]$
16 **end**
17 **end**
18 **for** $j \leftarrow 0$ to $i - 1$ **do**
19 **if** $\Lambda[j]$ has a higher height/id than $\Lambda[\Pi[j]]$ **then**
20 $\Pi[j] \leftarrow i$
21 **end**
22 **end**
23 Sort the merges of Λ by increasing height
24 **return** Λ

next step, and so on (see Figure 1). In this way, we can parallelize the merging phase of the algorithm, improving the overall efficiency. Moreover, the sooner we can combine dendrograms with overlapping datasets, the sooner we can eliminate duplicate or overlapping merges, reducing the total number of merges and the computational cost of the merging phase of the algorithm. In our implementation, we assign datasets $D_1 \cup D_2$ to the first thread, $D_1 \cup D_3$ to the second, and so on, so we combine the dendrograms belonging to consecutive threads first, as most consecutive pairs of threads share half of their datasets in common. (This is also illustrated in Figure 1.)

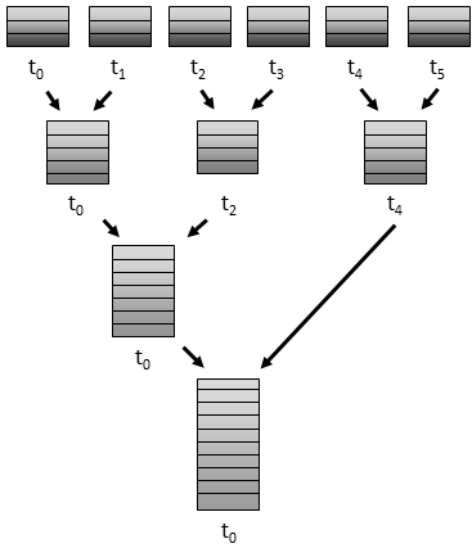


Fig. 1. Example for binary merging of 6 partial dendrograms (6 threads) in 3 rounds.

Algorithm 3: Pseudocode for merging two dendrograms

Input: M_1, M_2: Sorted merge lists of two dendrograms
Output: M_3: Sorted merge list of combined dendrogram

1 **Algorithm:** Merge(M_1, M_2)
2 Initialize M_3 to be empty
3 Initialize Union-Find data structure
4 Let (u_1, v_1, h_1) be the data points and height of the lowest merge in M_1
5 Let (u_2, v_2, h_2) be the data points and height of the lowest merge in M_2
6 **while** M_1 and M_2 have merges remaining **do**
7 **if** $h_1 < h_2$ **then**
8 **if** union(u_1, v_1) succeeds **then**
9 Add (u_1, v_1, h_1) to M_3
10 Let (u_1, v_1, h_1) be the next lowest merge in M_1
11 **else**
12 **if** union(u_2, v_2) succeeds **then**
13 Add (u_2, v_2, h_2) to M_3
14 Let (u_2, v_2, h_2) be the next lowest merge in M_2
15 **end**
16 **end**
17 Add the remaining merges (u, v, h) of M_1 or M_2 to M_3 as long as union(u, v) succeeds
18 **return** M_3

IV. Theoretical results

In this section, we discuss some of the theoretical aspects of our parallel algorithm, including its correctness, bounds on the amount of overhead introduced by our parallelization strategy, and algorithmic complexity. First, we formalize some of the language we will use for our theorems. A dendrogram is a relationship on a dataset that produces a partitional

clustering of the dataset for each height $h \in \mathbb{R}_{\geq 0}$. For any dendrogram, every distinct data point is in a separate cluster at height 0. We say that a SHC dendrogram merges two clusters $C_1 = \{x_1, x_2, \ldots, x_i\}$ and $C_2 = \{y_1, y_2, \ldots, y_j\}$ at height h iff $\min_{x \in C_1, y \in C_2}\{d(x, y)\} = h$ for some distance metric or dissimilarity measure d, and all vertices of $C_1 \cup C_2$ are considered to be in the same cluster for all $h' \geq h$. We also say that a dendrogram merges data points u and v at height h iff h is the minimum height at which u and v are considered to be in the same cluster. Note that the height at which a dendrogram merges u and v cannot be greater than $d(u, v)$.

A. Correctness

While we do not formally prove the correctness of the modified serial SLINK algorithm, we note that when the distance values are distinct, recording the points used to calculate the distance will produce the correct MST because SLINK calculates the correct SHC dendrogram, and, due to the way in which merges are compared in our modified algorithm, all distances are effectively unique. To formally establish the correctness of the parallel algorithm (Algorithm 1), we show that the dendrogram produced by Algorithm 1 merges every pair of points at the same height as the true dendrogram of the dataset. We need two lemmas to prove this result.

Lemma 4.1: If a dendrogram merges clusters containing data points u and v together at height h, then there exist some set of data points $x_1, x_2, \ldots, x_j$ such that $x_1 = u$, $x_j = v$, and the dendrogram merges data points x_i to x_{i+1} at height $d(x_i, x_{i+1}) \leq h$ for all such x_i. Further, $d(x_p, x_{p+1}) = h$ for some x_p.

Proof: We prove the claim by induction on the combined size of the clusters containing u and v. If the clusters containing u and v have two points between them, h must equal $d(u, v)$, so $x_1 = u$ and $x_2 = v$, and the claim is true trivially.

Suppose the claim is true when the two clusters have k or fewer points, and consider the case where the clusters contain $k + 1$ points. If $h = d(u, v)$, the claim holds as before. For the case where $h < d(u, v)$, there must be data points a and b such that a and u are in the same cluster, b and v are in the same cluster, and $h = d(a, b)$. Note that the cluster containing u and a must have k or fewer points, and the dendrogram must merge clusters containing u and a (with combined size k or less) at some height $h' \leq h$. Thus, by the inductive hypothesis, there must exist some set of points $x_1, x_2, \ldots, x_j$ such that $x_1 = u$, $x_j = a$, and the dendrogram merges x_i to x_{i+1} at height $d(x_i, x_{i+1}) \leq h' \leq h$. Similarly, there must exist some $y_1, y_2, \ldots, y_k$ such that $y_1 = v$, $y_k = b$, and the dendrogram merges y_i to y_{i+1} at height $d(y_i, y_{i+1}) \leq h$. Therefore, for the sequence of points $u = x_1, x_2, \ldots, x_j, y_k, y_{k-1}, \ldots, y_1 = v$, the dendrogram merges x_i to x_{i+1} at height $d(x_i, x_{i+1}) \leq h$, x_j to y_k at height $d(a, b) = h$, and y_i to y_{i+1} at height $d(y_i, y_{i+1}) \leq h$ for all i, proving the claim for clusters with total size $k + 1$. (In this case, $d(x_j, y_k) = h$.) ■

Lemma 4.2: If the SHC dendrogram of a dataset D merges clusters containing data points u and v together at height h, then the SHC dendrogram of a dataset $D' \supset D$ will merge data points u and v together at some height $h' \leq h$.

Proof: By Lemma 4.1, there exist some $x_1, x_2, \ldots, x_j$ in D such that $x_1 = u$, $x_2 = v$, and the dendrogram for D merges x_i and x_{i+1} at height $d(x_i, x_{i+1}) \leq h$ for all such x_i. Since $D \subset D'$, all of these x_i exist in D'. As a dendrogram must merge any two vertices a and b at a height no more than $d(a, b)$ and $d(x_i, x_{i+1}) \leq h$ for all x_i, the SHC dendrogram for D' must merge every x_i and x_{i+1} at a height no more than h, so x_i and x_{i+1} must be in the same cluster when the dendrogram of D' is cut at height h for all x_i. Thus, $x_1 = u$ and $x_j = v$ must be in the same cluster, so u and v are merged at some height $h' \leq h$. ■

We now present our main correctness result in Theorem 4.3. Essentially, we show that the merges in the optimal solution are all identified by the algorithm and propagated to the final solution, and the solution found by the algorithm cannot find a solution that merges data points at a lower height than the optimal solution.

Theorem 4.3: Let M be the true SHC dendrogram of the dataset, and let M' be the dendrogram calculated by Algorithm 1. For every pair of data points u and v, M' merges u and v at the same height as M.

Proof: As the SHC dendrogram must merge u and v at some height $h \leq d(u, v)$, we consider two cases.

(Case 1: The SHC dendrogram merges u and v at height $h = d(u, v)$.) As all pairs of data points are distributed among the parallel processes, at least one process p^* must be assigned a dataset containing both u and v. As we employ the SLINK algorithm on each processor, we know that p^* will calculate the correct SHC dendrogram for its partial dataset. We first show that this dendrogram must merge u and v at height h.

As p^* contains u and v and $d(u, v) = h$, the SHC dendrogram of p^* must merge clusters containing u and v at a height of at most h. Since the dataset assigned to p^* is a subset of the full data, Lemma 4.2 states that p^* must merge u and v at a height of at least h, so p^* must merge u and v at height exactly h.

We show by contradiction that no other dendrogram can merge u and v at height less than h. Suppose that some dendrogram merged u and v at a height $h' < h$. Thus, there must be some set of merges at height less than h that join u and v in the same cluster, so there must be some set of vertices $x_1 = u, x_2, \ldots, x_j = v$ such that the dendrogram merges x_i to x_{i+1} at height $d(x_i, x_{i+1}) < h$ for all such x_i. (Note that the height of these merges do represent the true distance between x_i and x_{i+1} as the merging procedure does not generate any new merges and the dendrogram for each partial dataset was calculated correctly.) As the all of these vertices exist in the full dataset, though, M should merge each x_i to x_{i+1} at height no more than than $d(x_i, x_{i+1}) < h$, so u and v should be in the same cluster at a height less than h; however, this contradicts the fact that M merges u and v at height h. Since no other dendrogram can merge u and v at height less than h, the merging process will conserve the merge between u and v at height h until all dendrograms have been merged, so M' will

also merge u and v at height $h = d(u, v)$.

(Case 2: The SHC dendrogram merges together clusters containing u and v at height $h < d(u, v)$.) By Lemma 4.1, there must exist some set of data points $x_1, x_2, \ldots, x_j \in C_1$ such that $x_1 = u$, $x_j = v$, and the SHC dendrogram merges clusters containing x_i and x_{i+1} at height $d(x_i, x_{i+1}) \leq h$ for all $i < j$. In addition, there must be some x_p such that x_p and x_{p+1} are merged together at height $d(x_p, x_{p+1}) = h$. By the result of Case 1, the final combined dendrogram will merge each x_i to x_{i+1} at height $d(x_i, x_{i+1}) \leq h$ and x_p to x_{p+1} at height h. Thus, all x_i (including u and v) must be in the same cluster at height h in the dendrogram formed by Algorithm 1. As the merging procedure of SHRINK would eliminate merges between u and v at a height greater than h, we focus on proving that the calculated dendrogram M' does not merge u and v at height less than h.

We prove by contradiction that u and v are merged together at a height no less than h in M'. Suppose u and v were merged at some height $h' < h$ in the dendrogram calculated by Algorithm 1. Thus, there must be some set of merges at height less than h that join u and v in the same cluster, so there must be some set of vertices $x_1 = u, x_2, \ldots, x_j = v$ such that the dendrogram merges x_i to x_{i+1} at height $d(x_i, x_{i+1}) < h$ for all such x_i. As the all of these vertices exist in the full dataset, though, M should merge each x_i to x_{i+1} at height no more than than $d(x_i, x_{i+1}) < h$, so u and v should be in the same cluster at a height less than h; however, this contradicts the fact that M merges u and v at height h.

As u and v are merged at height no more than h and no less than h in the final combined dendrogram, u and v are merged at a height of exactly h, proving the claim. ■

B. Bounding the amount of repeated computation

In SHRINK, every distance computation is performed by some thread, but the distances within a partition are calculated multiple times, resulting in repeated computation. In this section, we establish an upper bound on the total number of distance calculations performed by SHRINK. In the first case, we show that the parallel algorithm performs no more than twice as many distance computations as the serial algorithm in the case where the number of partitions evenly divides the size of the dataset. In the second, we show an upper bound of three times as many calculations when the number of partitions does not evenly divide the dataset; however, it may be possible to tighten this bound.

Theorem 4.4: When the number of partitions divides n evenly, the total number of distance calculations performed by Algorithm 1 is no more than $2\binom{n}{2}$, where $n \geq 2$ is the number of data points.

Proof: Let $\mathcal{D}(n, k)$ represent the total number of distance calculations performed by the parallel algorithm on k divisions of n data points. As k divides n, each of the $\binom{k}{2}$ processes has $2(n/k)$ points, so the total number of distance calculations is given by

$$\mathcal{D}(n,k) = \binom{2n/k}{2}\binom{k}{2} \tag{1}$$
$$= (2n/k)(2n/k-1)/2 \cdot k(k-1)/2 \tag{2}$$
$$= n(k-1)(2n-k)/(2k) \tag{3}$$
$$= (n/2)(2n+1-k-2n/k) \tag{4}$$

Taking the partial derivative with respect to k yields:

$$\frac{\partial \mathcal{D}}{\partial k} = n^2/k^2 - n/2 \tag{5}$$
$$0 = n^2/k^2 - n/2 \tag{6}$$
$$nk^2 = 2n^2 \tag{7}$$
$$k = \sqrt{2n} \tag{8}$$

(Note that k and n must both be positive.) As the partial derivative is positive for small k and negative for large k, the number of distance calculations reaches its maximum at $k = \sqrt{2n}$. Thus, the maximum number of distance computations performed by SHRINK is given by:

$$\mathcal{D}(n, \sqrt{2n}) = (n/2)(2n+1-\sqrt{2n}-2n/\sqrt{2n}) \tag{9}$$
$$= (n/2)(2n+1-2\sqrt{2n}) \tag{10}$$
$$= n(n-\sqrt{2n}+1/2) \tag{11}$$
$$\tag{12}$$

As $n - \sqrt{2n} + 1/2 < n - 1$ for $n \geq 2$, $\mathcal{D}(n, \sqrt{2n}) < n(n-1) = 2\binom{n}{2}$, proving the claim. ■

Theorem 4.5: The total number of distance calculations performed by Algorithm 1 is no more than $3\binom{n}{2}$, where n is the number of data points.

Proof: Let $\mathcal{D}(n, k)$ represent the total number of distance calculations performed by the parallel algorithm on k divisions of n data points. As each division contains at most $2\left\lfloor \frac{n}{k} \right\rfloor + 1$ points and there are $\binom{k}{2}$ processes,

$$\mathcal{D}(n,k) \leq \binom{2n/k+1}{2}\binom{k}{2} \tag{13}$$
$$\leq \frac{(2n/k+1)(2n/k)}{2} \cdot \frac{k(k-1)}{2} \tag{14}$$
$$\leq \frac{n(2n+k)(k-1)}{2k} \tag{15}$$
$$\leq (n/2)(2n-1+k-2n/k) \tag{16}$$

As the partial derivative of this expression with respect to k,

$$\frac{\partial \mathcal{D}}{\partial k} = n/2 + n^2/k^2, \tag{17}$$

is everywhere positive (n must be positive), the maximum for this upper bound will occur at the maximum possible number of divisions, $k = n$. Thus, the maximum number of distance calculations is no more than:

$$\max_k\{\mathcal{D}(n,k)\} \le \mathcal{D}(n,n) \le (n/2)(2n-1+n-2) \quad (18)$$
$$\le (n/2)(3n-3) \quad (19)$$
$$\le (3/2)n(n-1) \quad (20)$$
$$\le 3\binom{n}{2}, \quad (21)$$

proving the claim. ∎

An interesting consequence of the proof of Theorems 4.4 and 4.5 is that the amount of overhead (in terms of repeated distance calculations) is quadratic, and in fact decreases beyond a certain number of processes. In the extreme case, partitioning the dataset into sets of size one (using $\binom{n}{2}$ processes) results in no repeated distance calculations; however, the maximum value for the number of distance calculations requires more processes than could realistically be applied to a problem. For example, on a dataset of 20,000 points, the maximum number of distance computations would occur when using 19,900 cores, so the reduction in overhead is largely theoretical.

C. Complexity analysis

We analyze the parallel time and space complexity of SHRINK (Algorithm 1) as follows. The task of assigning n data points to p threads requires $O(n/\sqrt{p})$ time and space to enumerate the $2n/k$ data points, where k is the number of data partitions.

If we assume that the dataset has $O(1)$ dimensions, the time complexity of Algorithm 2 is $O(n_0^2)$, where n_0 is the number of data points passed to the algorithm. This time complexity is due to the **for** loop in line 5, which executes $O(n_0^2)$ times and has complexity $O(1)$. (The **for** loops in lines 8 and 18 both take $O(1)$ time, and the sort in line 23 takes $O(n_0 \log n_0)$ time.) Moreover, the space complexity is $O(n_0)$, as the three arrays allocated by the algorithm are of size $O(n_0)$ in addition to the input data. As each process has $2n/k$ data points, the parallel complexity for step 3 is $O(n^2/k^2)$ time and $O(n/k)$ space, or $O(n^2/p)$ time and $O(n/\sqrt{p})$ space.

Because we use a Union-Find data structure for step 6, we can compute the merged dendrogram in $O(n\alpha(n))$ time, where α is the inverse Ackermann function, due to the fact that the dendrograms may contain up to $O(n)$ total merges. The space complexity for this step is $O(n)$, due to the Union-Find data structure and the dendrograms themselves. By merging the dendrograms in a binary fashion, step 6 of Algorithm 1 will be repeated $O(\log p)$ times, giving a time complexity of $O(n \log p)$ (see Figure 1). Thus, the overall complexity for SHRINK is $O(n)$ space and $O(n^2/p + n/\sqrt{p} + n\alpha(n)\log p)$ time (or $O(n^2/p + n\alpha(n)\log p)$ if we assume $p < n^2$). By applying $p \sim n/\log n$ processes, we can reduce the time complexity of SHRINK to $O(n\alpha(n)\log(n))$.

V. Experimental Results

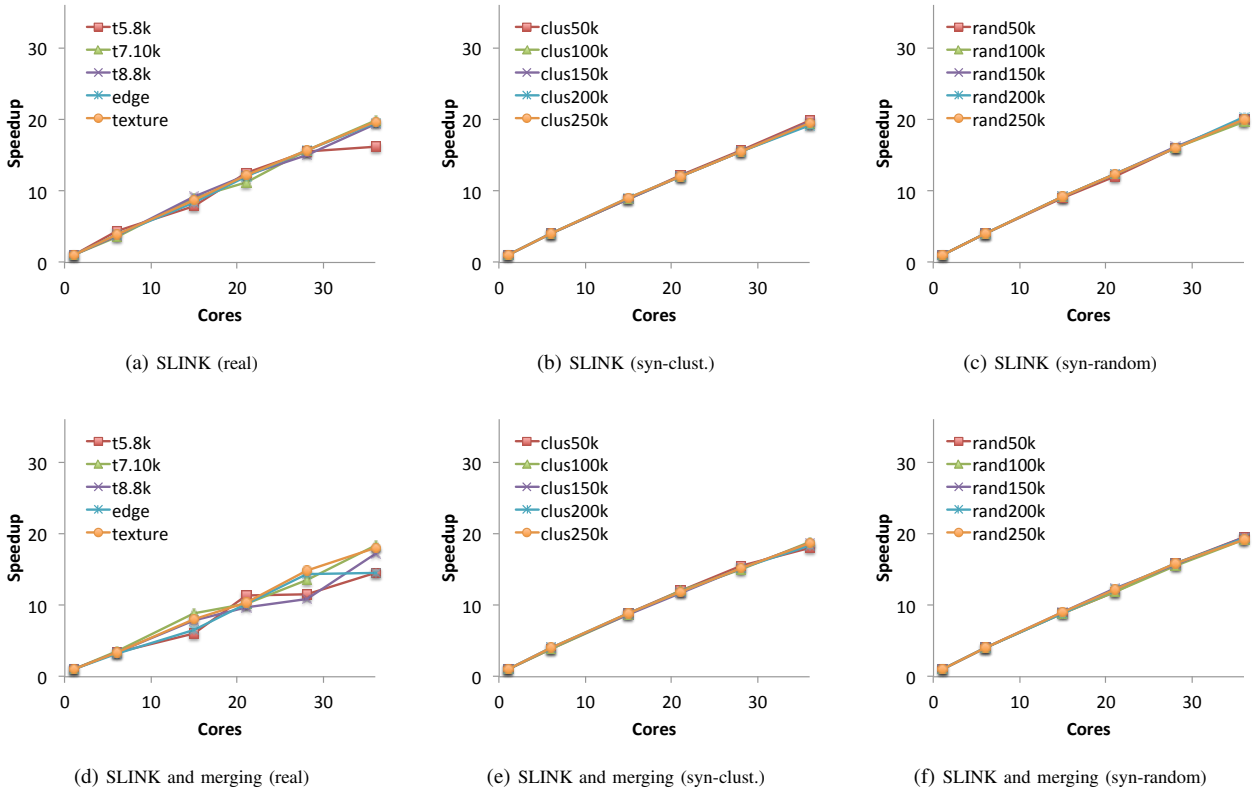
In this section, we present our empirical evaluation of SHRINK. As we are not aware of any openly available multicore codes for hierarchical clustering, we only evaluate the time and memory requirements of SHRINK. For these experiments, we used a Dell computer running GNU/Linux and equipped with four 2.00 GHz Intel Xeon E7-4850 processors with a total of 128 GB memory. Each processor has ten cores, each with 48 KB of L1 and 256 KB of L2 cache, and each processor shares a 24 MB L3 cache. All algorithms were implemented in C using OpenMP and compiled with gcc (version 4.6.3) using the `-O2` optimization flag.

Our testbed consists of 15 datasets divided into three categories, with five datasets each. The first category, called *real*, represent real-world data and have been collected from Chameleon (*t5.8k*, *t7.10k*, and *t8.8k*) [20] and CUCIS (*edge* and *texture*) [21]. The other two categories, *synthetic-random* and *synthetic-cluster*, have been generated synthetically using the IBM synthetic data generator [22], [23]. Whereas the data points in the synthetic-random datasets are drawn from a uniform random distribution, the synthetic-cluster datasets are generated by selecting random points cluster centers and then adding points to these clusters following a Gaussian distribution. The synthetic datasets contain 50 to 250 thousand points with 10 dimensions. Table I shows structural properties of the various datasets along with the the time taken by SHRINK, using one and thirty-six cores (t_1 and t_{36}, respectively). Though these results do not appear in the table, we also compared SHRINK against the SLINK implementation in the Cluster 3.0 library [18], and we found SHRINK to be faster than SLINK (up to twice as fast), likely due to the fact that we do not transform the dendrogram from *pointer representation* into *packed representation*, as well as the reduced amount of pre- and postprocessing.

TABLE I
STRUCTURAL PROPERTIES OF THE TESTBED (REAL, SYNTHETIC-CLUSTER, AND SYNTHETIC-RANDOM) AND THE TIME TAKEN BY SHRINK USING ONE AND THIRTY-SIX PROCESS CORES (t_1 AND t_{36}, RESPECTIVELY). NOTE THAT t_1 TYPICALLY OUTPERFORMED THE SLINK IMPLEMENTATION IN THE CLUSTER 3.0 LIBRARY [18], LIKELY DUE TO OUR MODIFICATION TO THE SLINK ALGORITHM (ALGORITHM 2)

			Time (seconds)	
Name	Points	Dimensions	t_1	t_{36}
$t5.8k$	24,000	2	6.47	0.445
$t7.10k$	30,000	2	9.41	0.512
$t8.8k$	24,000	2	5.96	0.346
$edge$	17,695	18	6.72	0.464
$texture$	17,695	20	7.04	0.389
$clust50k$	50,000	10	36.16	2.02
$clust100k$	100,000	10	147.09	7.84
$clust150k$	150,000	10	341.92	18.36
$clust200k$	200,000	10	609.30	33.21
$clust250k$	250,000	10	953.65	50.99
$rand50k$	50,000	10	42.95	2.21
$rand100k$	100,000	10	174.55	9.10
$rand150k$	150,000	10	397.08	20.42
$rand200k$	200,000	10	710.49	36.60
$rand250k$	250,000	10	1,124.49	58.68

Figure 2 shows the speedup obtained by SHRINK (Algorithm 1) for various numbers of process cores (threads). The raw runtime taken by SHRINK on one process core



(a) SLINK (real) (b) SLINK (syn-clust.) (c) SLINK (syn-random)

(d) SLINK and merging (real) (e) SLINK and merging (syn-clust.) (f) SLINK and merging (syn-random)

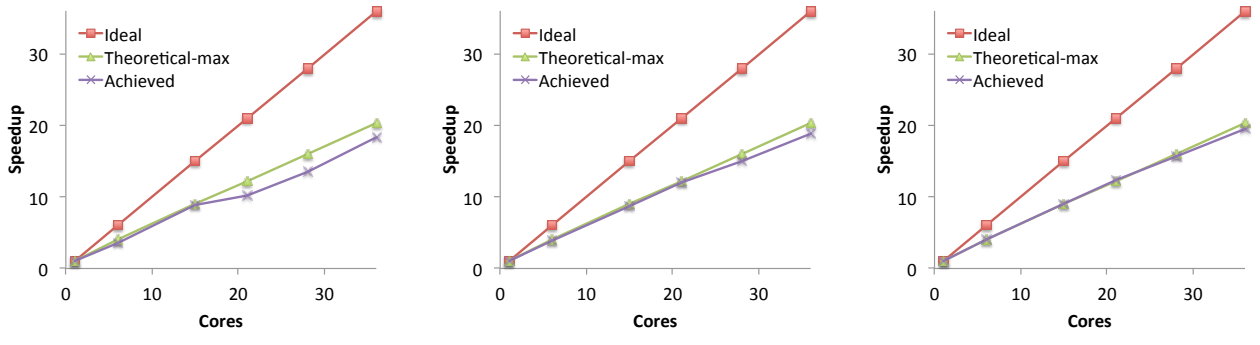
Fig. 2. Speedup of SHRINK. Top row: Computation time of modified SLINK algorithm in SHRINK. Bottom row: Total time (SLINK + merging) of SHRINK.

appears in Table I. The top row in Figure 2 shows the speedup obtained by considering only the time taken by the modified SLINK algorithm (Algorithm 2), whereas the bottom row shows results using the total time (SLINK plus merging) for the three datasets.

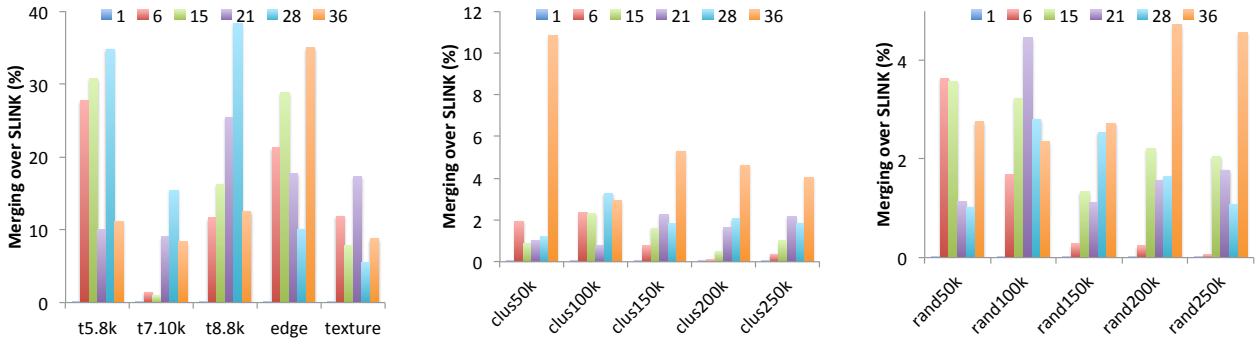
As discussed in Section IV-B, SHRINK incurs a cost of some repeated distance computations as the number of threads increases. While the amount of overhead depends on both the data size and the number of threads, for sufficiently large datasets, the influence of the data size is quite small. In particular, for datasets between 17,695 and 250,000 points, the amount of redundant computation on 36 threads ranges between 77.74% and 77.78% of the distance calculations performed in the serial case, leading to a theoretically maximum speedup of 20.250 to 20.254. In light of the overhead induced by our approach, the speedup results for the SLINK phase is nearly ideal for all cases. Moreover, since the time taken by the merging phase was quite short (less than a second for all datasets with less than 200k points), the speedup behavior of the SLINK phase is very similar to that of the overall execution, though it does affect the speedup results for the real datasets, which were much smaller than our synthetic datasets.

Figure 3 presents a more detailed analysis of the speedup results for SHRINK. The top row in Figure 3 shows a comparison between ideal speedup, theoretical maximum, and the achieved speedups for the best dataset from each of the three categories, real, synthetic-cluster, and synthetic-random. As noted earlier, the theoretical maximum approaches one half of ideal scaling due to the redundant distance computations, though our speedup results match these values very closely for all three datasets. The bottom row in Figure 3 shows a comparison between the amount of time spent in the merging phase relative to the SLINK phase. While the merging time was short in all cases, it formed a larger fraction of the total time on the smaller datasets, up to 40% of the time spent on the SLINK phase, though this fraction decreases substantially for the larger synthetic datasets. While many of the cases show an overall trend towards a longer merging time for more cores (as would be expected), there are several exceptions, possibly due to the exact distribution of the data (i.e., how many merges could be eliminated early on in the merging phase).

Figure 4 presents an analysis on the memory used by our implementation of SHRINK. The top row shows how the memory requirements change with the data size and number of threads. (Since the cores are running SLINK and performing merges independently, the memory requirements increase with the number of threads.) As a point of comparison, a triangular distance matrix for 250,000 points would require more than 100 GB to store. The bottom row shows a breakdown of the memory used by variable for one of the datasets from each category. The variables are as follows: `data` stores the full dataset, `rowid` stores the IDs for the data points

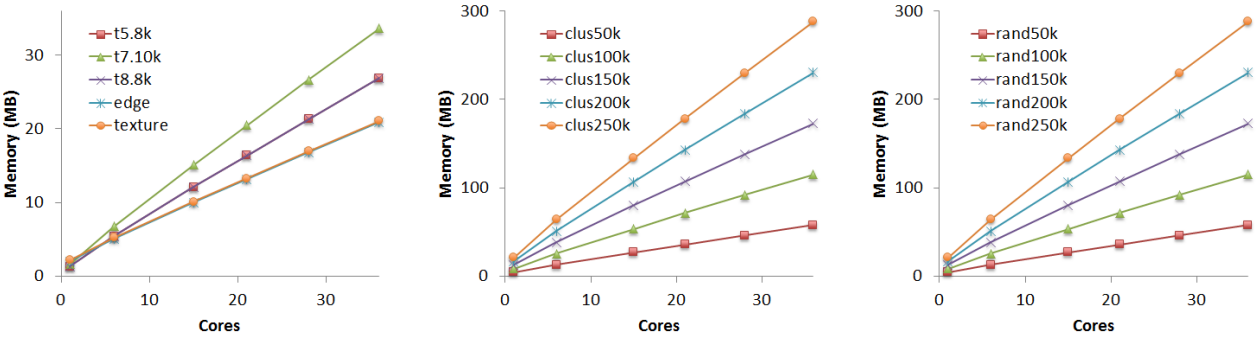


(a) Comparing speedups (*t7.10k*) (b) Comparing speedups (*clust100k*) (c) Comparing speedups (*rand150k*)

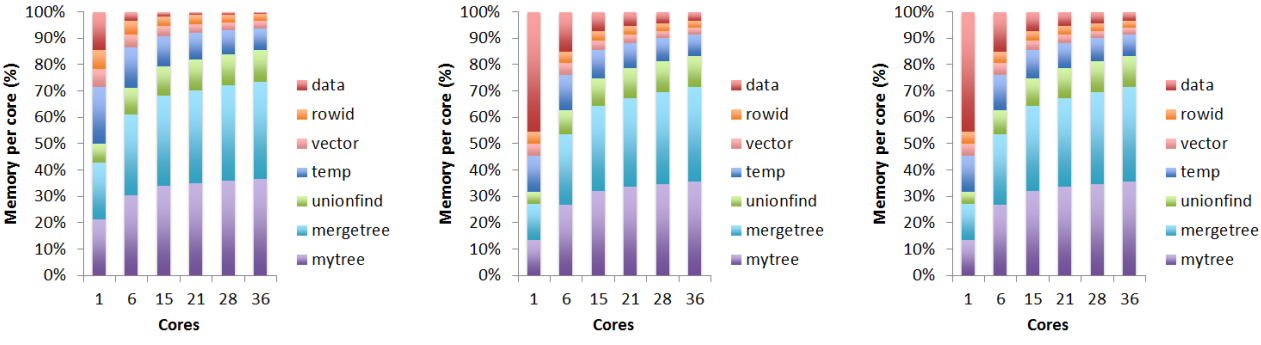


(d) Merging over SLINK (real) (e) Merging over SLINK (syn-clust.) (f) Merging over SLINK (syn-random)

Fig. 3. Top row: Comparing the ideal, theoretical maximum, and the achieved speedups for three datasets (*t7.10k*, *clust100k*, and *rand150k*), one from each category. Bottom row: Comparing the time taken by merging over SLINK in percentage for varying number of threads.



(a) Required memory (real) (b) Required memory (syn-clust.) (c) Required memory (syn-random)



(d) Memory distribution (*t7p10k*) (e) Memory distribution (*clust100k*) (f) Memory distribution (*rand250k*)

Fig. 4. Top row: Memory requirement in MB for varying number of threads. Bottom row: Detailed memory breakdown for three datasets (*t7.10k*, *clust100k*, and *rand150k*).

that belong to each thread, `mytree` stores the thread-local dendrogram, `mergetree` is a buffer for merging two dendrograms, `unionfind` is the Union-Find data structure used for merging dendrograms, and `temp` and `vector` are arrays used by the SLINK algorithm (M and Π, respectively). As the number of threads increase, `data` remains constant; `rowid`, `temp`, and `vector` are allocated for each thread but decrease individually as the data partitions shrink; and `unionfind`, `mytree`, and `mergetree` are allocated for each thread but remain the same size. Thus, we can see that the memory requirements become dominated by the dendrogram and the Union-Find data structure as the number of threads increases.

VI. Conclusion

In this paper, we have described SHRINK, a parallel algorithm for single-linkage hierarchical clustering. We also investigated some of the theoretical properties of SHRINK, including a bound on the amount of duplicate work and a formal proof of correctness. We also evaluated SHRINK empirically, finding that it achieves speedup within 10% of our predicted optimal performance across a majority of our evaluated datasets. Lastly, we have made the code for SHRINK openly available for download at http://cucis.ece.northwestern.edu. Future work on SHRINK may involve efforts to extend SHRINK to a hybrid OpenMP-MPI platform to take advantage of multi-processor, multicore environments. Additionally, we may try to tighten the theoretical bounds on the complexity or the amount of repeated work. Finally, the parallelization strategy employed by SHRINK may be applicable to other types of problems, particularly those that can be modeled as dense graph problems.

Acknowledgments

This work is supported in part by NSF award numbers CCF-0621443, OCI-0724599, CCF-0833131, CNS-0830927, IIS-0905205, OCI-0956311, CCF-0938000, CCF-1043085, CCF-1029166, and OCI-1144061, and in part by DOE grants DE-FG02-08ER25848, DE-SC0001283, DE-SC0005309, DE-SC0005340, and DE-SC0007456.

References

[1] A. El-Hamdouchi and P. Willett, "Comparison of hierarchic agglomerative clustering methods for document retrieval," *The Computer Journal*, vol. 32, no. 3, pp. 220–227, 1989.

[2] P. Tamayo, D. Slonim, J. Mesirov, Q. Zhu, S. Kitareewan, E. Dmitrovsky, E. Lander, and T. Golub, "Interpreting patterns of gene expression with self-organizing maps: Methods and application to hematopoietic differentiation," *Proceedings of the National Academy of Sciences*, vol. 96, no. 6, pp. 2907–2912, 1999.

[3] J. Xu and A. Hagler, "Chemoinformatics and drug discovery," *Molecules*, vol. 7, no. 8, pp. 566–600, 2002.

[4] R. Sibson, "Slink: An optimally efficient algorithm for the single-link cluster method," *The Computer Journal*, vol. 16, no. 1, pp. 30–34, 1973.

[5] J. L. Bentley, "A parallel algorithm for constructing minimum spanning trees," *Journal of Algorithms*, vol. 1, no. 1, pp. 51–59, 1980.

[6] C. Olson, "Parallel algorithms for hierarchical clustering," *Parallel Computing*, vol. 21, no. 8, pp. 1313–1325, 1995.

[7] M. Dash, S. Petrutiu, and P. Scheuermann, "Efficient parallel hierarchical clustering," in *Euro-Par 2004 Parallel Processing*, ser. Lecture Notes in Computer Science, M. Danelutto, M. Vanneschi, and D. Laforenza, Eds. Springer Berlin / Heidelberg, 2004, vol. 3149, pp. 363–371.

[8] D. Chang, M. Kantardzic, and M. Ouyang, "Hierarchical clustering with cuda/gpu." in *ISCA PDCCS*, J. Graham and A. Skjellum, Eds. ISCA, 2009, pp. 7–12.

[9] Z. Du and F. Lin, "A novel parallelization approach for hierarchical clustering," *Parallel Computing*, vol. 31, no. 5, pp. 523–527, 2005.

[10] R. Cole, P. Klein, and R. Tarjan, "Finding minimum spanning forests in logarithmic time and linear work using random sampling," in *Proceedings of the eighth annual ACM symposium on Parallel algorithms and architectures*, ser. SPAA '96. New York, NY, USA: ACM, 1996, pp. 243–250.

[11] C. Poon and V. Ramachandran, "A randomized linear work erew pram algorithm to find a minimum spanning forest," in *Algorithms and Computation*, ser. Lecture Notes in Computer Science, H. Leong, H. Imai, and S. Jain, Eds. Springer Berlin / Heidelberg, 1997, vol. 1350, pp. 212–222.

[12] S. Pettie and V. Ramachandran, "A randomized time-work optimal parallel algorithm for finding a minimum spanning forest," *SIAM Journal on Computing*, vol. 31, no. 6, p. 1879, 2002.

[13] S. Chung and A. Condon, "Parallel implementation of boruvka's minimum spanning tree algorithm," in *Parallel Processing Symposium, 1996., Proceedings of IPPS '96, The 10th International*, apr 1996, pp. 302–308.

[14] D. A. Bader and G. Cong, "A fast, parallel spanning tree algorithm for symmetric multiprocessors (smps)," *Journal of Parallel and Distributed Computing*, vol. 65, no. 9, pp. 994–1006, 2005.

[15] D. Bader and G. Cong, "Fast shared-memory algorithms for computing the minimum spanning forest of sparse graphs," *Journal of Parallel and Distributed Computing*, vol. 66, no. 11, pp. 1366–1378, 2006.

[16] F. Dehne and S. Götz, "Practical parallel algorithms for minimum spanning trees," in *Reliable Distributed Systems, 1998. Proceedings. Seventeenth IEEE Symposium on*, oct 1998, pp. 366–371.

[17] V. Olman, F. Mao, H. Wu, and Y. Xu, "Parallel clustering algorithm for large data sets with applications in bioinformatics," *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, vol. 6, pp. 344–352, April 2009.

[18] M. de Hoon, S. Imoto, J. Nolan, and S. Miyano, "Open source clustering software," *Bioinformatics*, vol. 20, no. 9, pp. 1453–1454, 2004.

[19] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. The MIT Press, 2009.

[20] "Cluto - clustering high-dimensional datasets," 2006, http://glaros.dtc.umn.edu/gkhome/cluto/cluto/.

[21] "Parallel k-means data clustering," 2005, http://users.eecs.northwestern.edu/ wkliao/Kmeans/.

[22] R. Agrawal and R. Srikant, "Quest synthetic data generator," *IBM Almaden Research Center*, 1994.

[23] J. Pisharath, Y. Liu, W. Liao, A. Choudhary, G. Memik, and J. Parhi, "Nu-minebench 3.0," Technical Report CUCIS-2005-08-01, Northwestern University, Tech. Rep., 2010.