DIMACS Series in Discrete Mathematics
and Theoretical Computer Science
Volume 00, 0000

# Finding clusters quickly in parallel

BRENDAN MUMEY

January 27, 1995

ABSTRACT. This report describes the implementation of a new parallel algorithm for finding clusters point sets of Euclidean space. A new solution to a special case of the *bichromatic closest pairs problem* is used by the algorithm. The implementation was done in an environment for distributed tree computations on the Kendall Square Research KSR-2 multiprocessor.

## 1. Introduction

A popular method in the astrophysics community for determining the clustering present is the *friends-of-friends* approach. It is based on the bottleneck distance metric. The bottleneck distance between two points is defined to be the minimum possible weight of the maximum weight edge in a path between them:

$$d_b(x,y) = \min_{\text{paths } P \text{ connecting } x \text{ and } y} \max_{\text{edges } e \in P} \text{length}(e)$$

That is, if you only charge according to most lengthy edge traversed then the bottleneck cost is the cost of the least expensive path between a pair of points. Two points in a tightly clustered clump will be close under the bottleneck distance metric as there will be many nearby intermediate points to go through. Only the maximum length edge matters. Conversely, points which are isolated must incur large bottleneck costs to reach other points. Two points $x$ and $y$ are *$\tau$-clustered* if and only if $d_b(x,y) \leq \tau$. The object is, for a given $\tau$, to find all the $\tau$ clusters. One way of doing this is to first construct a Euclidean minimum spanning tree (EMST) on the point set and then remove all edges which are longer than $\tau$. The remaining connected components constitute the $\tau$ clusters. Unfortunately constructing an EMST in three dimensions is a difficult problem for the problem size of interest, $n > 10^6$. The best sequential algorithm seems to be Yao's $O((n \lg n)^{1.5})$ time method[**Y82**].

1991 *Mathematics Subject Classification.* Primary 68Q22, 68Q25; Secondary 68-04.

A natural compromise is to find a good approximation to the exact solution. We say an approximate cluster finding algorithm is $\epsilon$-*close* if:

(i) $\delta_b(x,y) > (1+\epsilon)\tau \Rightarrow x$ and $y$ are identified to be in different clusters

(ii) $\delta_b(x,y) \leq \tau \Rightarrow x$ and $y$ are identified to be in the same cluster

For many astrophysics clustering problems, $\epsilon$ as high as 0.1 is acceptable according to the practitioners the author has asked.

This papers presents a new parallel algorithm to find $\epsilon$-close $\tau$ clusters. The worst case time is $O((\lg \frac{\Delta}{\epsilon\tau} + f(\epsilon))n/p)$ in the PRAM model, where $\Delta$ is the diameter of the point set and $p$ is the number of processors. The function $f$ depends on the dimension of the point set. For two dimensions we can show $f(\epsilon) = O(\sqrt{1/\epsilon})$ and for three dimensions we can show $f = O(1/\epsilon)$ although we conjecture a somewhat better bound (Appendix A.1). We begin we an overview of the algorithm and go on to describe its implementation for the Kendall Square KSR-2 shared memory multiprocessor and performance achieved on astrophysical data.

## 2. The Algorithm

We present the serial version of the algorithm first. The issues related to its parallelization are described subsequently. The algorithm is divided into two phases. In the first phase an *oblivious kd-tree* is constructed on the point set. This is an adaptation of standard kd-trees [**B75**]. A bounding box for the point set is first found. We will assume that the sidelengths of this bounding box are equal (this is needed later in the proof of the running time bound). In general, the length of the largest dimension present in the current box is split exactly in half to form two child boxes. The process is repeated for each non-empty child box recursively. When the length of the longest dimension is less than $\theta = \epsilon\tau/4$, the box is no longer split. The value of $\theta$ is chosen so that estimating the maximum (minimum) distance between two points in $\theta$-sized boxes by the maximum (minimum) distances between the box boundaries meets the approximation bounds above. The tree constructed is said to be $\theta$-*resolved*. The maximum depth of a $\theta$-resolved oblivious kd-tree is $O(\lg(\Delta/\epsilon\tau))$. It is easy to establish that the cost of building the tree is proportional to the depth of the tree times $n$, so the time cost of the first phase of the algorithm is $O(\lg(\Delta/\epsilon\tau)n)$.

The second phase of the algorithm is more complicated. The basic idea is to start at a point $x$ and determine which points are contained within a ball centered on $x$ of radius $\tau$. These points can be added to the cluster containing $x$. If a ball is expanded around every point it is fairly easy to argue that all the clusters will be found correctly: If two points $x$ and $y$ have a bottleneck distance less than the threshold $\tau$, then there will be a path through points in the point set connecting them such that the distances between adjacent points in the path are all less than $\tau$. Since we expand a ball around every point we will merge

together all the points in this path, including $x$ and $y$ into the same cluster.

The cluster-finding phase of the algorithm walks through the tree in a depth-first fashion, starting from the root. If the cluster of the current tree cell has already been determined nothing is done and the cell is not further divided. If not, and bounding box associated with the current cell has diameter at most $\tau$, then all of its particles are immediately marked as belonging to the same cluster a local search is conducted to see if there are particles with distance $\tau$ of the cell to merge into this cluster. This is accomplished by searching back up the tree to locate the neighboring cells and testing each in turn to find new points to merge into the cluster. A subroutine is used to check whether two cells contain points which belong to the same cluster. It uses a filtering technique to reduce the work. The subroutine is described below and an analysis of its running time can be found in Appendix A. In the bounding box for the current cell has diameter greater than $\tau$ the tree walk continues to the left child cell.

**2.1. The Comparison-Pair Subroutine.** Given two nearby boxes in the kd-tree, both having diameter less than $\tau$, the problem is to determine whether they belong to the same cluster. This will happen if and only if there is some particle in the first box, call it $L$, which is within distance $\tau$ from some particle in the second box, call it $R$. This is a special case of the bichromatic closest-pairs problem discussed in [**KI92**].

Let $m$ be the total number of points in both boxes. We present an $O(f(\epsilon) \cdot m)$ time method for this special case. Bounds on $f$ are determined in Appendix A. The basic idea is to keep a queue of box-box pairs. The idea is that the queue will contain pairs of boxes which might contain particles which link $L$ and $R$, but further checking is necessary. The queue is initialized to the pair containing the left and right box. New *comparison-pairs* are generated as follows. First the pair at the head of the queue is removed. Each box in the pair is subdivided into its two sub-boxes and the cross-product of the left two sub-boxes with right two sub-boxes is formed. For each pair in the cross-product, the maximum and minimum separations between the box boundaries are calculated. If the maximum separation is at most $\tau + \epsilon$, we know that pair will contain a pair of particles separated by at most $\tau + \epsilon$ and we can merge the $L$'s cluster with $R$'s immediately. If the minimum separation is greater than $\tau$ then this particular pair cannot possibly contain points that link $L$ and $R$ and so may be ignored. Any pairs remaining in the cross-product are added to the end of the queue for further checking. These pairs have minimum boundary separation at most $\tau$ and maximum boundary separation greater than $\tau + \epsilon$. The next comparison-pair is removed from the head of the queue and the above process is repeated. This continues until a comparison-pair is found which links $L$ and $R$ or the queue is empty.

**2.2. Parallelization Issues.** The first major obstacle to confront in implementing this algorithm in parallel is the fact that the central data structure, the

kd-tree must be distributed. If there is a single address space (as in the KSR-2) this is easy, but if not some means of locating parts of the tree residing in other processor's address spaces must be provided. If the tree is to be constructed in parallel the underlying point set needs to be distributed among the processors first and they must collaborate in the building process. The approach we take is to have one processor determine the top of the tree and then let each processor build one of the subtrees. The top part of the tree is duplicated in each processor and the nodes associated the subtree roots indicate in which processor that subtree resides.

A more complicated issue arises in the cluster-building phase. As clusters may cross processor boundaries, some means of linking clusters across these boundaries must exist. This is accomplished with a global naming scheme for clusters. A cluster name is a tuple `(iPart, id)`, where `iPart` is the number of the cluster's founding particle and `id` is the processor number where this particle is located.

Furthermore, as clusters are found in parallel and may need to be merged, some means of merging clusters must be available. We use a locked version of the standard *union-find* data-structure [**CLR**] for this purpose. Of course some means of locking must be implemented if not provided.

## 3. Implementation and Performance

This section describes the current implementation of the algorithm for the KSR-2 and its performance on real data. The implementation differs from the the algorithm presented in that a standard kd-tree is used, rather than an oblivious kd-tree. For most distributions, it was thought that performance would be comparable. The standard kd-tree has the advantage that it is balanced and thus requires fewer pointers and is generally simpler to implement. Since space seems to be the limiting resource, this decision seemed appropriate. The implementation also has the ability to handle *quasi-periodic* boundary conditions, a feature we do not describe.

The program was written on top of a system for managing distributed data structures called *MDL*. MDL was developed by Joachim Stadel in the Astrophysics department at the University of Washington. The idea of MDL (machine dependent layer) is to provide a common interface for accessing shared data across processors so the higher levels of the program are machine-independent. To port to a different multiprocessor, one just needs to rewrite the MDL part and recompile. In general each processor has a thread which sends and receives messages from similar threads on the other processors. This mechanism is used to access parts of the kd-tree which reside on nonlocal processors. These accesses include both reads and writes. There are functions for returning a local pointer to a node's data, given an `iCell` and `id`, returning the parent and children of a node, etc. The provision for locking a node is also provided. This is neces-

sary for merging clusters, as discussed above. Although the KSR-2 has a shared address space we make use of it only minimally and expect to port to purely message-passing architectures.

The implementation performed quite well. Figures 1 and 2 summarize the performance achieved by the KSR-2 on a $10^6$ particle file representing a globular cluster of galaxies from an astrophysical simulation [1]. The values of $\tau$ and $\epsilon$ were taken to be 10.0 and 0.1 respectively; typical numbers for this data set. Experiments on different particle distributions were conducted to ensure the one chosen was representative; this was the case. It is somewhat curious that the speed-up curve does not have steadily decreasing slope. Two explanations are offered: First, the KSR-2 used had two rings of 32 processors each. Inter-ring communication is about twice as slow as intra-ring communication; this would reduce efficiency most dramatically when most processors are on one ring but a couple are on the other. Another possible explanation that this is a data distribution effect. The tree which gets built depends on the number of processors used. There may be some variation in the efficiency of the tree which is built.

The reason for diminishing speed-up returns is that that proportion of nonlocal work increases with the number of processors used. Roughly speaking, this is because nonlocal work work is proportional to the surface area of the region of space assigned to a processor, whereas local work is proportional to the volume. This ratio increases with more processors. In contrast to the PRAM model, the cost of nonlocal memory accesses is at least a factor of 10 more than the cost of local accesses. In this respect, a theoretical model which takes this phenomenon into account (eg. *LogP* [**LOGP**]) would be more predictive. Inevitably it is more complicated to prove time bounds in such a model.

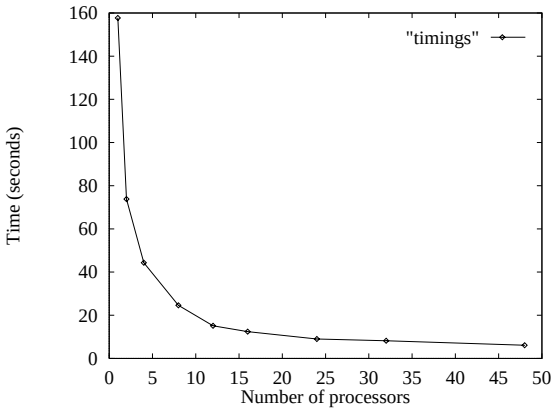


FIGURE 1. KSR-2 timings for a $10^6$ particle globular cluster

[1] The author is grateful to R. Carlberg, Department of Astrophysics, University of Toronto for providing test data sets.

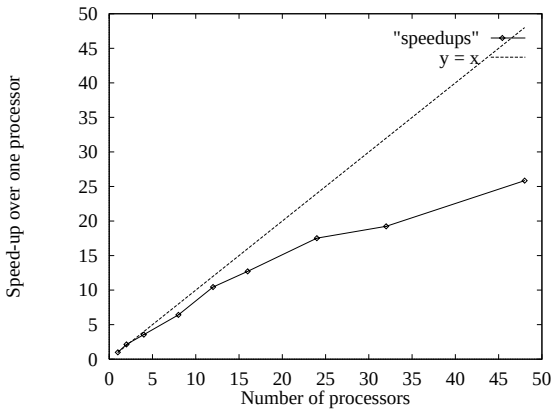


FIGURE 2. Plot of KSR-2 speed-ups

## 4. Conclusions

We have developed a new parallel friends-of-friends cluster finder and have achieved good performance with a shared memory implementation on the Kendall Square KSR-2 multiprocessor. The implementation is current being used by the theoretical astronomy group at the University of Washington. The algorithm uses a new solution to a special case of the bichromatic closest pairs problem which may be of interest in its own right.

## REFERENCES

[B75] J. Bentley, *Multidimensional binary search trees used for associative searching*, Comm. of the ACM **18** (1975), 509–517.

[CLR] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to algorithms*, MIT Press, Cambridge, 1990.

[LOGP] D. Culler, R. Karp, D. Patterson, A. Sahay, K. Schauser, E. Santos, R. Subramonian, and T. von Eicken, *LogP: Towards a realistic model for parallel computation*, Proc. 5th Symp. on Parallel Algorithms and Architectures (1993).

[KI92] N. Katoh and K. Iwano, *Finding k farthest pairs and k closest/farthest bichromatic pairs for points in the plane*, Proc. 8th ACM Symp. on Computational Geometry (1992).

[Y82] A. Yao., *On constructing minimum spanning trees in k-dimensional spaces and related problems*, SIAM Journal on Computing **11** (1982), 721–736.

## Appendix A. Analysis of the comparison-pair subroutine

The following lemma gives an upper bound on the running time of a particular invocation of the comparison-pair subroutine. We present the proof in two dimensions and mention how it generalizes to three dimensions.

LEMMA A.1. *Let the comparison-pair subroutine be invoked on two boxes, L and R, which contain a total of m points between them. For two dimensional point sets, the subroutine finishes in* $O(\sqrt{1/\epsilon} \cdot m)$ *time.*

Before we can prove the main lemma, we need some definitions and several

auxiliary lemmas. Let $P$ be the set of comparison-pairs examined during the execution of the subroutine. The time spent by the algorithm is proportional to $|P|$.

When the comparison-pair subroutine is initially invoked, the left and right boxes have the same dimensions. This is because because the bounding boxes at any given level in an oblivious kd-tree have the same shape. A technicality occurs when one of the boxes consists of singleton point. Rather than taking using exact position in the calculations, we will assume that there is a hierarchy of smaller boxes surrounding it. The sizes of these boxes will correspond to the box sizes in the tree at the levels below the singleton box. In this way, comparisons-pairs will always be formed by boxes at the same level in the tree. Thus we can partition the set $P$ by the tree level of each pair. In the context of the proof, we will say the initial comparison-pair on which the subroutine is invoked has level 0. It makes the analysis easier if we associate two spheres with each box $i$. Let $c_i$ be the center of box $i$. We define $R_l$ to be the radius of the smallest sphere which can enclose boxes at level $l$. Similarly $r_l$ is defined to be the radius the largest sphere which fit inside boxes at level $l$. Note that $R_{l+2} = R_l/2$ and $r_{l+2} = r_l/2$. Since we are assuming the initial bounding box of the point set is chosen so that sidelengths are all equal, it easy to establish that for all $l \geq 0$,

$$R_l/r_l \leq \sqrt{2} \tag{A.1}$$

and

$$2^{-(l/2+3)}\tau < R_l \leq 2^{-(l/2+1)}\tau. \tag{A.2}$$

Let $P_l$ be the set of comparison-pairs at level $l$ examined by the subroutine. We have

$$P = \bigcup_{l=0}^{2\lg(4/\epsilon)} P_l.$$

If $(i,j) \in P_l$ we observe that

$$\tau - 2R_l < |c_i - c_j| < \tau + 2R_l. \tag{A.3}$$

(If the first inequality fails, this pair of boxes will be known to contain points to link $L$ and $R$, and so the subroutine would have terminated. If the second inequality fails, the pair is separated too much to possibly contain such points.)

The strategy for bounding $|P|$ will be to bound each of $|P_l|$ in terms of the number of non-empty sub-boxes of $L$ and $R$ at level $l$. We make the observation that the total number of non-empty sub-boxes at level $l$ is at most $m$. Let the *neighbors of box $i$* be the set

$$N(i) = \{j : (i,j) \in P \text{ or } (j,i) \in P\}.$$

From now on we will interchangeably refer to comparison-pairs as edges and non-empty box centers as vertices. For each level $l \geq 0$, define a value $K_l$ by

$$K_l = \frac{32\sqrt{\tau R_l}}{\pi r_l}.$$

We will charge each edge $(i,j) \in P_l$ to one or two vertices according to the following rule:

| | | |
|---|---|---|
| If | $\lvert N(i)\rvert \leq K_l$ | charge vertex $c_i$ one, |
| else if | $\lvert N(j)\rvert \leq K_l$ | charge vertex $c_j$ one, |
| else | | charge vertex $c_i$ and $c_j$ one-half. |

Denote the set of vertices at level $l$ by $V_l$. It follows that

$$|P_l| = \sum_{c_x \in V_l} \text{charge}(c_x). \tag{A.4}$$

LEMMA A.2. *For all vertices $c_x \in V_l$, charge$(c_x) < K_l$.*

Consider a vertex $c_i \in V_l$. If $|N(i)| \leq K_l$, charge$(c_i) \leq K_l$, so the only case to consider is when $N(i) > K_l$. Let $J = \{j : j \in N(i) \text{ and } N(j) > K_l\}$. We have charge$(c_i) = |J|/2$. We will show that $|J| > 2K_l$ leads to a contradiction and so charge$(c_i) \leq K_l$. To do this we need a couple more lemmas.

LEMMA A.3. *If $|J| > 2K_l$ then there exists vertices $\{c_j, c_k, c_l\}$ such that:*
(i) *$c_k, c_l \in J$ and $c_j \in N(l)$*
(ii) *$|c_i - c_j|, |c_k - c_l| > \sqrt{8\tau R_l}$*
(iii) *Edge $(c_j, c_l)$ crosses edge $(c_i, c_k)$*

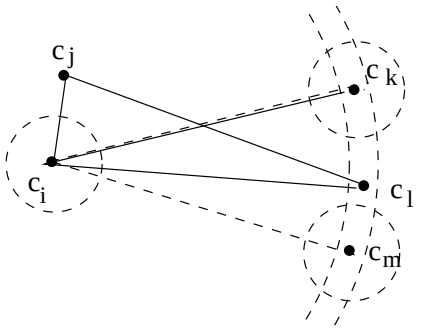


FIGURE 3. A possible configuration of the vertices $\{c_i, c_j, c_k, c_l\}$.

PROOF. Since $J \subset N(i)$, by ( A.3) we have $\tau - 2R_l < |c_i - c_x| < \tau + 2R_l$ for each $x \in J$. Thus $c_x$ falls in an annulus around $c_i$ of thickness $4R_l$ (see Figure 3). The set $J$ exists in some section of this annulus. Let $c_k$ and $c_m$ be the extremal points bounding this section. Each vertex $c_x \in J$ has a ball around it of radius

$r_l$ which cannot contain other vertices. Within distance $\sqrt{8\tau R_l}$ of $c_k$ or $c_m$, the volume available for points of $J$ is less than $2 \cdot (8R_l \cdot \sqrt{8\tau R_l})$. Less than

$$\frac{2 \cdot 8R_l \cdot \sqrt{8\tau R_l}}{\pi r_l^2} \leq 2 \cdot K_l$$

points can fit in this space. Hence there must be a point $c_l \in J$ with $|c_l - c_k| > \sqrt{8\tau R_l}$ and $|c_l - c_m| > \sqrt{8\tau R_l}$. We know that $|N(l)| > K_l$ and $c_i \in N(l)$. Similar reasoning shows there must exist a point $c_j \in N(l)$ with $|c_i - c_j| > \sqrt{8\tau R_l}$. The edge $(c_j, c_l)$ must cross one of the edges $(c_i, c_k)$ or $(c_i, c_m)$. Without loss of generality, it crosses $(c_i, c_k)$ (relabel $c_k$ and $c_m$ if necessary). The vertices $\{c_j, c_k, c_l\}$ have all properties listed in the lemma. □

LEMMA A.4. *Such a configuration of vertices is impossible in Euclidean space.*

PROOF. Referring back to Figure 3, we have the following constraints:

(i) $\tau - 2R_l < |c_i - c_k|, |c_i - c_l|, |c_j - c_l| < \tau + 2R_l$
(ii) Edge $(c_j, c_l)$ crosses edge $(c_i, c_k)$
(iii) $|c_i - c_j|, |c_k - c_l| > \sqrt{8\tau R_l}$

We show the first two constraints forbid the third. Assume the first two constraint hold. From the first constaint, the point $c_l$ must fall in the intersection of annuli of thickness $4R_l$ centered at $c_i$ and $c_j$ respectively. As indicated in Figure 4, the point $c_k$ must be located somewhere above the point $c_l$ in order to maintain the second constraint. The first contraint also implies that $c_k$ must fall in the annulus centered at $c_i$. We also know that $|c_j - c_k| > \tau - 2R_l$ at this point, because the subroutine would have terminated (by linking $L$ and $R$) previously while processing comparison-pairs at level $l - 1$. This contrains $c_k$ to lie in the shaded region shown in the figure. Thus $|c_k - c_l|$ is bounded above by the distance between the points where the outer boundary of one annulus meets the inner boundary of the other annulus. Let this distance be $q$. Some simple trigonometry shows that $|c_i - c_j| \cdot q = 8\tau R_l$, so

$$|c_i - c_j| \cdot |c_k - c_l| \leq |c_i - c_j| \cdot q = 8\tau R_l.$$

But the second constraint implies that

$$|c_i - c_j| \cdot |c_k - c_l| > 8\tau R_l.$$

This contradiction completes the proof of this lemma, which in turn completes the proof of Lemma 2. □

PROOF OF LEMMA 1: We make the observation that for any level $l$, $|V_l| \leq m$. (Note the $V_l$ includes the "imaginary" boxes around singletons, introduced to ensure comparison-pairs consist of boxes at the same level.) This is easy to see: If for some level $l$, $|V_l| > m$, there would be more than $m$ pairwise-disjoint, non-empty boxes in the tree at this level and $L$ and $R$ would have to contain more

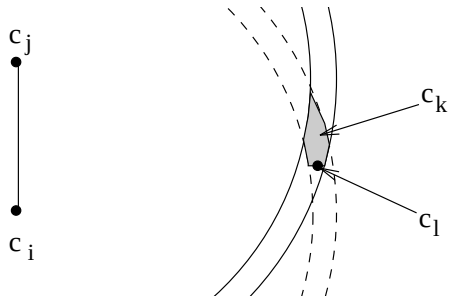


FIGURE 4. Locations of $c_k$ and $c_l$ permitted by constraint one.

than $m$ points. We have:

$$\begin{aligned} |P| &= \sum_{l=0}^{2\lg(4/\epsilon)} |P_l| = \sum_{l=0}^{2\lg(4/\epsilon)} \sum_{c_x \in V_l} \text{charge}(c_x) \\ &< \sum_{l=0}^{2\lg(4/\epsilon)} \sum_{c_x \in V_l} K_l < m \sum_{l=0}^{2\lg(4/\epsilon)} K_l. \end{aligned}$$

Using (A.1) and (A.2) we derive $K_l \leq \frac{256}{\pi} 2^{l/4}$. Evaluation of the $K_l$ sum above gives

$$|P| \leq 1025\sqrt{1/\epsilon} \cdot m.$$

As previously remarked, the running time of the subroutine is $O(|P|) = O(\sqrt{1/\epsilon} \cdot m)$.

**A.1. Three dimensions.** A proof similar to the above can given in three dimensions. For $K_l = O(2^{l/3})$ lemmas analogous to Lemmas 2, 3 and 4 can be shown. The recurrence relation on $R_l$ is now $R_{l+3} = R_l/2$. This changes the upper limit in the $K_l$ sum above to $3\lg(4/\epsilon)$. The bound on $|P|$ becomes $O((1/\epsilon) \cdot m)$. The author conjectures a tighter bound of perhaps $O((1/\epsilon)^{1.5})$ on the size of $|P|$, but has not established this result at the time of writing.

DEPARTMENT OF COMPUTER SCIENCE, UNIVERSITY OF WASHINGTON, SEATTLE, WASHINGTON, 98195

*E-mail address*: brendan@cs.washington.edu