Design and Evaluation of a Parallel HOP Clustering Algorithm for Cosmological Simulation

Ying Liu, Wei-keng Liao, Alok Choudhary

Department of Electrical and Computer Engineering
Northwestern University
Evanston, IL 60208
{yingliu, wkliao, choudhar}@ece.northwestern.edu

Abstract

Clustering, or unsupervised classification, has many uses in fields that depend on grouping results from large amount of data, an example being the N-body cosmological simulation in astrophysics. In this paper, we study a particular clustering algorithm used in astrophysics, called HOP, and present a parallel implementation to speed up its current sequential implementation. Our approach first builds in parallel the spatial domain hierarchical data structure, a three-dimensional KD tree. Using a KD tree, the core of the HOP algorithm that searches for the highest density neighbor can be performed using only subsets of the particles and hence the communication cost is reduced. We evaluate our implementation by using data sets from a production cosmological application. The experimental results demonstrate up to 24× speedup using 64 processors on three parallel processing machines.

1. Introduction

The size of various data sets has increased tremendously in recent years as speedups in processing and communication have greatly improved the capability for data generation and collection in areas such as scientific experimentation, business and government transactions, as well as the Internet. Traditional knowledge discovery systems have been found lacking in their ability to handle current data sets, due to characteristics such as their large sizes and high dimensionality. Consequently, new techniques that can automatically transform these large data sets into useful information are in strong demand. Data mining, which is the process of discovering useful and understandable patterns hidden in massive data sets, has become an important research area. Clustering is one of the important data mining techniques whereby the data set is partitioned into subsets containing elements of similar properties. Due to the huge size and high dimensionality of the available data sets, it is quite common to see databases on the order of gigabytes or terabytes. A sequential clustering algorithm handling these large data sets would potentially be unable to run in-core or would take a tremendous amount of time. Therefore, parallel computing is an essential component of the solution to speed up large size data clustering.

The HOP algorithm is a hierarchical clustering algorithm in astrophysics [1]. Having assigned to every particle an estimate of its local density, HOP associates each particle with the densest neighbor of its N_{hop} nearest particle neighbors. By repeating this process, a hopping path can be generated for each particle in the direction of increasing density. The path ends when it reaches a particle that is its own densest neighbor. All particles reaching the same such particle are identified as a group. HOP is spatially adaptive, coordinate-free, and the results are insensitive to most of the subjective user input parameters.

In this paper, we present a parallel implementation of the HOP clustering algorithm. In order to achieve load balance, our approach distributes the particles evenly across all processors by constructing a parallel KD tree. The inter-processor data communication is implemented by combining many small size communication requests into a single large request. By transferring the required particles beforehand, searching for the highest density neighbor can be performed locally in each processor. This strategy is also applied to the process of hopping and group merging. We ran our experiments on three parallel machines using the data outputs from a production cosmological simulation application, called ENZO. Two sets of star particle data, each with different spatial distribution pattern, were tested. The experimental results present up to 24× speedup using 64 processors. Our parallelization approach is also applicable to other scientific fields, such as molecular biology, geology, or astronomy as long as they have similar requirements on clustering and neighbor finding procedures.

The rest of this paper is organized as follows. The next section overviews the related works of clustering algorithms. Section 3 describes the HOP algorithm. Our design and parallelization of HOP algorithm is presented in Section 4. Software development issues are given in Section 5. Section 6 presents our experimental results and we conclude the paper in Section 7.

2. Related work

Data mining, also referred as knowledge discovery, is a process of extracting implicit, previously unknown and potentially useful information. It is commonly used in many fields, including scientific computation, financial analysis, information retrieval, decision making, and the World Wide Web. Data mining techniques can be roughly categorized into classification, clustering, association rules, sequence mining, similarity search, and other methods. Classification is a method that assigns a new record to one of several predefined categories or classes. Clustering, also referred as unsupervised classification, is to group a set of data (without a predefined class attribute) based on the conceptual clustering principles: maximizing the intra-class similarity and minimizing the interclass similarity. This technique is useful in document retrieval, image segmentation, taxonomy generation, biomedical engineering, geological spatial data analysis, earth science, telecommunication, etc. Clustering algorithms can be further divided into two categories: agglomerative algorithms and partitional algorithms. Agglomerative algorithms find clusters by initially assigning each object to its own cluster and then repeatedly merging pairs of clusters until a certain stopping criterion is met [5]. Examples include group average [6], single-link [7], CURE [8], ROCK [9] and CHAMELEON [10]. Partitional algorithms find clusters by partitioning the entire dataset into either a predetermined or an automatically derived number of clusters. Examples include k-means [11], k-medoids [6], auto-class [12] and graph-partitioning-based method [13].

Clustering algorithms are commonly applied in astrophysics, such as cosmological simulations that identify collapsed halos, determine the merge of two galaxies and locate dwarf galaxies. IsoDen [4], FOF [17], DENMAX [18] and HOP [1] are clustering algorithms proposed and used in cosmological simulation. IsoDen finds clusters by constructing an adaptive oct-tree with individual particles stored in the leaves while internal nodes represent the cubical regions of space. The idea of FOF is that two particles belong to the same group if their separation is less than some chosen threshold and, then, such pairs are chained into groups. However, some groups found by FOF may appear as two clumps linked by a small thread of particles between the subgroups that may be inappropriate for some applications [1]. DENMAX takes a different approach by first estimating the density of each particle and then, determining the group of the particle through tracing path along the gradient of its density surface until a local maximum is reached. All particles that end up at the same maximum are defined to the same group. The drawback is that it may result in splitting large groups into smaller pieces due to the discovery of multiple local maxima at finer resolution [1]. HOP [1] is a clustering algorithm that overcomes the drawbacks of FOF and DENMAX.

Since most of clustering applications are computational and data intensive, many efforts have been contributed to their parallelization recently. Examples of parallel agglomerative methods are SLINK algorithm [14], Prim's minimal spanning tree algorithm [15], Ward's minimum variance method [16], etc. An example of a partitional method is parallel K-Means [19]. Several works have been applied to cosmological simulation, such as Halo World [4], a parallel implementation of IsoDen method in which a parallel tree library handles all the communication on message passing parallel architecture.

3. HOP clustering algorithm

Instead of constructing the density gradient field in FOF and DENMAX, HOP assigns a density to each particle and replaces the concept of gradient by a simple search within the particle's neighborhood to find the neighbor with the highest density. Each particle is associated to its highest density neighbor. Through this densest neighbor, a particle can "hop" to the next highest density particle and continues hopping until it reaches a particle that is its own highest density neighbor. All particles that hop to the same particle constitute a single group. Groups that share a sufficiently dense boundary are reconnected. Particles whose densities are less than a given threshold are excluded from groups.

The HOP clustering algorithm consists of four processing stages:

1) ***Constructing a KD tree*:** To reduce the workload of locating N_{dens} nearest neighbors for each particle, we use a KD tree structure to partition the particles based on their spatial locations, so that nearby particles are in the same sub-domain. Therefore, the neighborhood searching process can be performed by first checking the sub-domain's boundary before going through all the particles in it. K-dimensional (KD) tree, a data structure used in orthogonal range searching, is commonly used in domain partition application [3]. KD tree is a balanced tree that partitions the spatial domain recursively along the longest axis into sub-domains. Each sub-domain

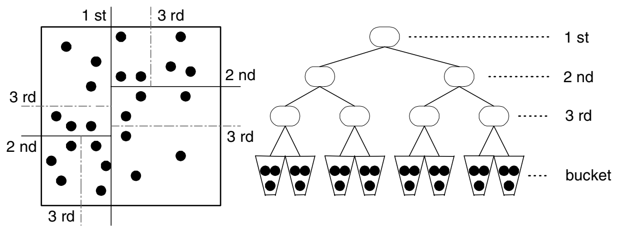


Fig. 1. Two-dimensional KD tree with 24 particles. Bucket size is 3 and the KD tree has 3 levels. Every internal node contains the boundary information of its sub-domain. Bisection is performed by the median coordinate value of the particles in the sub-domain.

contains approximately the same number of particles. The root node represents the entire simulation domain that covers all the particles and each tree node represents a sub-domain of its parent node. In a KD tree, only the leaf nodes (also called buckets) contain the particle data. The less number of particles in a bucket, the more levels of the tree depth. Every sub-domain partitioning involves determining the longest axis and searching the median coordinate value among all particles along that axis. One key property of a KD tree is that particles spatially closed are located in the same buckets or sibling buckets of the same tree branch. This property makes it very efficient to find neighbors of any particle. Figure 1 illustrates an example of a two-dimensional KD tree. In HOP, *bucket size* (the number of particles in a bucket) is a user-provide parameter that determines the tree depth and affects the computational cost of the neighborhood search in the next stage.

2) ***Generating density:*** HOP estimates the density of a particle by using an adaptive kernel with a length scale set by the distance to its N_{dens} nearest neighbor particles, where N_{dens} is a user-provide parameter. HOP employs SMOOTH as its density generating mechanism. SMOOTH [2] is a KD-tree based near neighbor search engine that can calculate some mean quantities for particles in an N-Body simulation, such as mean velocity, mean speed, and velocity dispersion. This stage is the most computational intensive part of HOP due to large amount of computation when searching for N_{dens} neighbors.

3) ***Hopping*:** The core of HOP is to associate each particle to its highest density neighbor within its N_{hop} nearest neighbors, where N_{hop} is a user-provide parameter ($N_{hop} <= N_{dens}$). A particle continues "hopping" to a higher density until it reaches the particle that is its own highest density neighbor. Since the hopping is performed in increasing density order, its convergence is guaranteed.

4) ***Grouping:*** Particles associated to the same densest particle are defined as a group. Therefore, every particle is assigned to one and only one group. To refine the clustering quality, group merging and pruning are performed according to some chosen density thresholds. Groups can be merged if their boundary satisfies some thresholds. Particles whose densities don't exceed the density thresholds are excluded from the groups. A group might be disbanded if its maximum density is less than a given density threshold.

4. HOP parallelization

The key idea of our parallel implementation is to distribute the data particles across all processors evenly to ensure the balanced computational workload and access remote particles through communication. Thus, the balanced load and the data locality contribute to the efficiency of the parallelization. We assume the initial state of the parallel HOP as following: given an array of N particles, each of P processors reads N/P particles exclusively. The particles owned by a processor can locate arbitrarily across the whole problem domain. The data structure of a particle consists of its mass value and three spatial coordinates.

4.1 Constructing a parallel KD tree

Starting from the root-node of the KD tree, we first determine the longest axis d and, then, find the median value m of all particles' d coordinates in parallel. The whole spatial domain is bisected into two sub-domains by m. Particles are exchanged between processors such that the particles whose d coordinate are greater than m go to one sub-domain and the rest particles to the other one. Therefore, an equal number of particles are maintained in each sub-domain after the bisection. We repeat this procedure recursively in every sub-domain till the number of sub-domains is equal to the number of processors. Then, each processor continues to build its own local tree

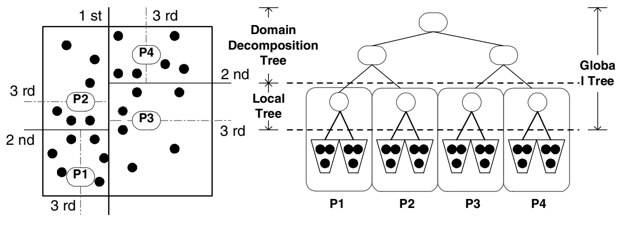


Fig. 2. Two-dimensional KD tree distributed over four processors. Each processor contains 6 particles. Bucket size is 3 and the global tree has 3 levels. Local tree can be built concurrently without communication. Every processor maintains the same copy of the global tree.

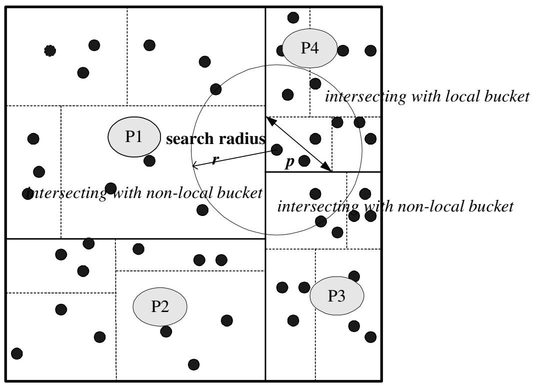

P4
intersecting with local bucket
P1
search radius
r
p
intersecting with non-local bucket
intersecting with non-local bucket
P2
P3

Fig. 3. Intersection test for particle *p* in processor P4. The neighborhood is a sphere region with a search radius *r*. The neighborhood search domain of particle *p* intersects with P1 and P3.

within its domain until the desired bucket size (number of particles in each leaf) is reached. Note that inter-processor communication is not required when constructing the local trees.

We maintain a copy of the whole tree on every processor so that the communication overhead incurred at performing search domain intersection test with the remote local trees at the stages 2 and 3 can be reduced. Therefore, at the end of this stage, local trees are broadcasted to all processors. As shown in Figure 2, the root-node of the KD tree represents the entire simulation domain while each of the rest tree nodes represent a rectangular sub-domain of its parent node. The information contained in a non-leaf tree node includes the aggregated mass, center of mass, number of particles, and domain boundaries. When the KD tree is completed, particles are divided into spatially closed regions of approximately equal number. The advantage of using a KD tree is not only its simplicity but also the balanced data distribution.

4.2 Generating density

The density of a particle is estimated by its N_{dens} nearest neighbors, where N_{dens} is a user-provide parameter. Since it is possible that some of the N_{dens} neighbors of a particle are owned by remote processors, communication is required to access non-local neighbor particles at this stage. Our approach is, for each particle, first to perform intersection test by traversing the global tree with a given initial search radius *r* while keeping track of the non-local intersected buckets, as shown in Figure 3. If the total number of particles in all intersected buckets is less than N_{dens}, the intersection test is re-performed with a larger radius. Once tree walking is completed for all local particles, all the remote buckets containing the potential neighbors are obtained through communication. Note that there is only one communication request to each remote processor to gather the intersected buckets. No further communication is necessary when searching for its N_{dens} nearest neighbors. Since the KD tree displays the value of spatial locality, particle neighbors are most likely located in the same or nearby buckets. According to our experimental results, the communication volume is only 10% - 20% of the total number of particles. However, with highly irregular particle distribution, communication costs may increase.

To calculate the density for particle *p*, we use a PQ tree (priority queue) [24] to maintain a sorted list of particles that are currently the N_{dens} nearest neighbors. The root of the PQ tree contains the neighbor farthest from *p*. If a new neighbor whose distance to *p* is shorter than the root, replace the root with the second farthest one and update the PQ tree. Finally, the particles remained in the PQ tree are the N_{dens} nearest neighbors of *p*.

4.3 Hopping

This stage first associates each particle to its highest density neighbor among its N_{hop} nearest neighbors that are already stored in the PQ tree generated at the previous stage. Each particle, then, hops to the highest density neighbor of its associated neighbor. Hopping to remote particles is performed by first keeping track of all the remote particles and, then, making a communication request to the owner processors. This procedure may repeat several times until all the needed non-local particles are already stored locally. Since the hopping is in density increasing order, the convergence is guaranteed.

4.4 Grouping

Particles linked to the same densest particle are defined as a group. However, some groups should be merged or refined according to the chosen density thresholds. Thus, every processor first builds a boundary matrix for the groups constructed from its local particles

Table 1. Parallel Machine Specification

Configuration / Machine	# Processor/Node	Memory/Node	Processor Type	Inter-connection
IBM SP2	8	SIMD, 12 GB	375 MHz, Power3	Colony Switch
SGI Origin2000	1	SIMD, 0.25 – 0.6 GB	195 MHz, 250 MHz, MPIS R1000	Gigabit Ethernet
Linux Cluster	2	MIMD, 512 MB	500 MHz, Pentium III	Myrinet

and, then, exchanges the boundary matrix among all processors. Particles whose densities are less than a given threshold are excluded from groups and two groups are merged if their boundary satisfies some given thresholds.

5. Software development

We develop our parallel implementation in C and use Message Passing Interface (MPI) [23] for the inter-processor communication. The program was developed in a Single Program Multiple Data (SPMD) model. We conducted our experiments on three parallel machines: an IBM SP2 at San Diego Supercomputing Center, an SGI Origin2000 at National Center for Supercomputing Application (NCSA), and a Linux Cluster at Argonne National Lab. System configurations are summarized in Table 1. We run our experiments using 1 to 64 processors on each of them.

6. Experimental results

To evaluate our parallel implementation of the HOP algorithm, we use the star particle data sets generated from a large-scale production cosmological application, ENZO, developed at NCSA [20], [21]. ENZO is a three-dimensional parallel application that simulates the formation of a cluster of galaxies consisting of gas and stars [21]. The simulation starts near the beginning of the universe, a few hundred million years after the big bang, when the galaxy is in a relative uniform radiation distribution, and continues till the present day, when it is in a highly irregular star particle distribution. It is used to test theories of how galaxy and clusters of galaxies form by comparing the results with what is really observed in the sky today [22]. The ENZO's outputs are periodical data dumps that show the evolution of the galaxy formation. In our experiments, we use two attributes of star particles: the Euclid coordinates in three dimensions and particle mass. The performance results were obtained by running our parallel HOP implementation on two output data sets from ENZO: one is in the earlier stage of the galaxy formation and the other is at the end of the simulation. Both data sets contain 491,520 star particles. Figure 4 shows the spatial distribution of these two data sets in three dimensions.

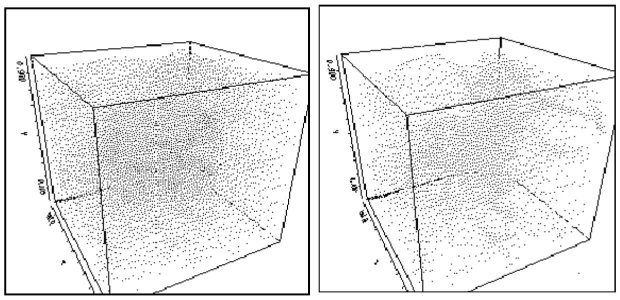

(a) Data set 1 (b) Data set 2

Fig. 4. 3-dimensional spatial distribution of two particle data sets from the ENZO outputs. Data set 1 represents the results at the earlier stage of the simulation. Data set 2 represents the more irregular distribution near the end of the simulation.

6.1 Performance evaluation

Throughout our experiments, we vary the following two parameters:

1). *bucket size*: It is the number of particles contained in each leaf of the 3-dimensinal KD tree. The larger the *bucket size*, the less number of the tree depth.
2). N_{dens}: A particle's density is estimated by the mass of its N_{dens} nearest neighbor particles. A larger N_{dens} means more computation and communication involved in locating the neighbors.

Although there are other parameters that can be varied, we only consider these two because they affect over 90% of the clustering performance. We also vary the number of processors from 1 to 64 to study the scalability of our implementation. Figure 5 presents the measured total execution time with break down time of each stage. The corresponding speedups are presented in Figure 6.

We observed that the overall performance scales up on both IBM SP2 and SGI Origin2000. However, on the Linux Cluster, the performance scales well till using 32 processors and the speedup goes down when using more processors. The reason is that the more processors are used, the less number of particles each individual processor owns, which results in relative lower computation cost and higher communication overhead. As we expected, data set 2 takes longer time than data set 1 in all cases since the more irregular the particle distribution is, the more inter-processor communication occurs.

We now examine the execution time for each individual stage. As observed in Figure 5, the density generation is the most time consuming stage, which takes more than 60 % of the total time. Figures 7 and 8 give the measured time and the speedups of this stage, respectively, which show similar performance curves as in Figures 5 and 6.

In order to find out what is the main cause of the speedup saturation, we now study the communication time and the computation time separately. From Figure 9 we can see that the communication time doesn't scale well, even increases when using more than 32 processors. On the other hand, from Figure 10 we can observe that the computation part of our approach obtains very good scalability. It obtains up to 45x speedup when using 64

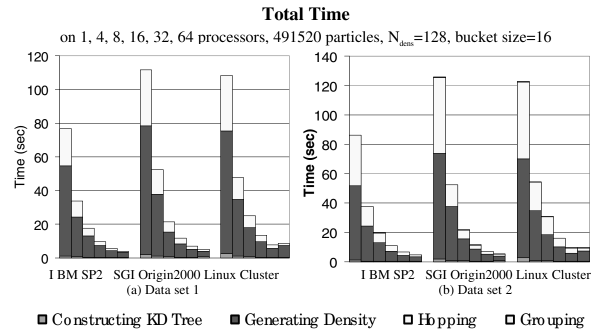


Fig. 5. Total execution time on IBM SP2, SGI Origin2000 and Linux Cluster.

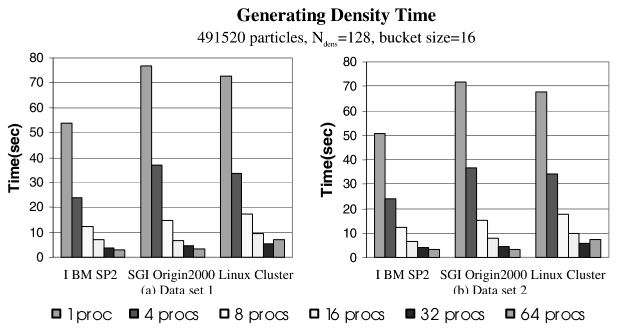


Fig. 7. Generating density time on IBM SP2, SGI Origin2000 and Linux Cluster.

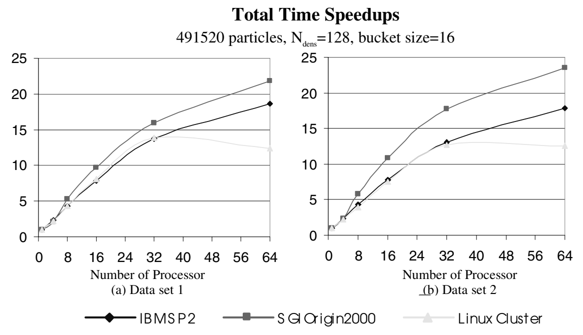


Fig. 6. Speedups of the total execution time on IBM SP2, SGI Origin2000 and Linux Cluster.

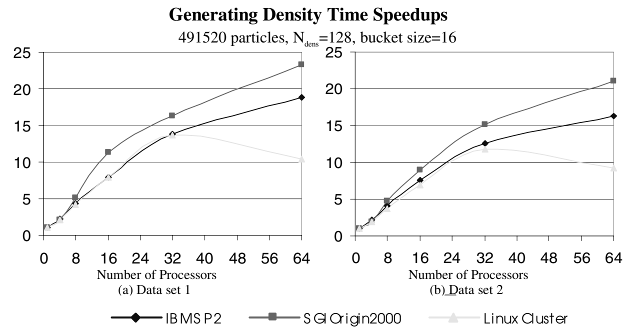


Fig. 8. Speedups of generating density on IBM SP2, SGI Origin2000 and Linux Cluster.

processors, which is much better than the overall speedup. Therefore, we can conclude that the communication overhead is the main cause of the speedup saturation.

KD tree construction is the first stage of the HOP parallelization. Particle transfer during the recursive bisection may result in a large amount of data communication. Particles can be moved multiple times before they go to their final destination processors. Figure 11 and Figure 12 present the execution time of this stage and the corresponding speedups, respectively. The speedups saturate when the number of processors goes beyond 32.

Varying *bucket size*. Varying the number of particles in each bucket affects the depth of the KD tree. Although a larger *bucket size* can reduce the time for constructing the parallel KD tree, it increases the total volume of particle transferring at the density generation stage and the hopping stage. Figure 13 shows the performance trend when varying *bucket size* from 16 to 128 using 1 to 64 processors. We observe that the execution time goes up with larger *bucket size* for most of the cases.

Varying N_{dens}. Figure 14 provides the performance of using 16, 32, 64 and 128 as N_{dens}. Although execution time increases when using a larger N_{dens} as more neighbor searching computation occurs, we get an interesting observation that the speedups improve when we increase N_{dens}. The reason is that the cost of communication is not increased as much as the computation.

In general, our experiments present good overall performance results of our parallel HOP implementation. We obtain up to 24x speedup using 64 processors.

7. Conclusions

We presented our parallel implementation of the HOP clustering algorithm. The parallel implementation distributes particle data across processors to guarantee the balanced computational load. The data structure used by the clustering algorithm is a KD tree that contributes to the efficiency of searching for neighbors. Our strategy is to obtain non-local data through an inter-processor communication before each individual processor can work independently. We benchmarked our implementation using data generated from a real cosmological simulation, ENZO. Two data sets with different spatial particle distributions were used in our test. Our experiments presented good speedups on different parallel machines, which benefit from the balanced data distribution and the minimal data communications.

Although HOP is a clustering algorithm in cosmological N-body problem, it may find applications in other fields, such as molecular biology, geology and

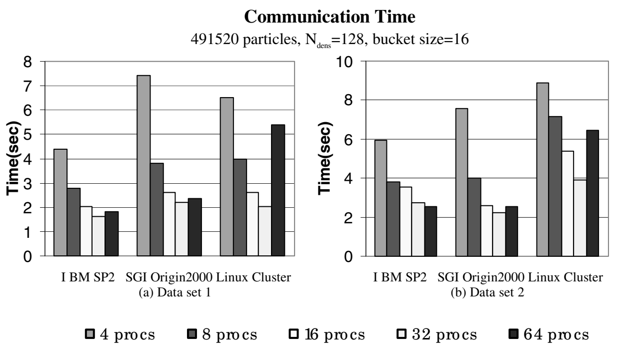


Fig. 9. Communication time on IBM SP2, SGI Origin2000 and Linux Cluster.

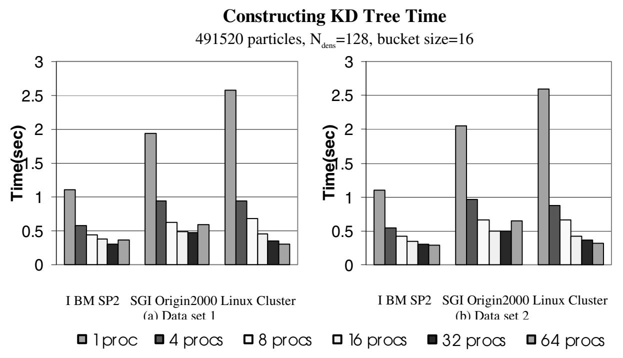


Fig. 11. Constructing KD tree time on IBM SP2, SGI Origin2000 and Linux Cluster.

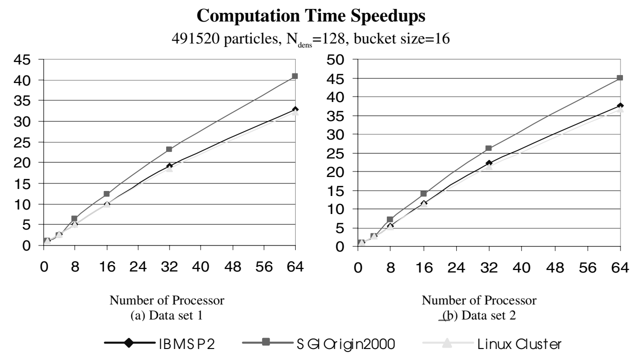


Fig. 10. Speedups of computation time on IBM SP2, SGI Origin2000 and Linux Cluster.

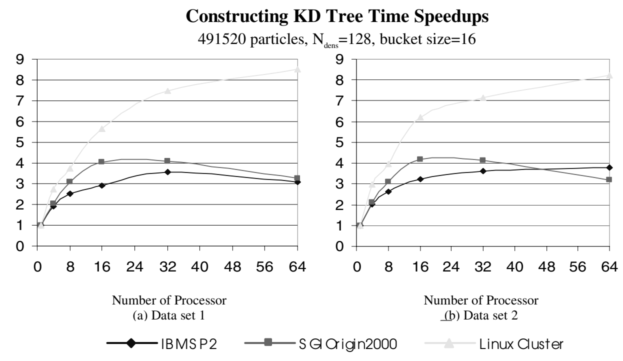


Fig. 12. Speedups of constructing KD tree time on IBM SP2, SGI Origin2000 and Linux Cluster.

astronomy, where large spatial data sets are to be processed with similar clustering or neighbor finding procedures. One example is molecular pattern recognition, which normally involves huge amount of neighbor finding computation. Our parallelization strategy is applicable to those fields.

Acknowledgements

This work is supported by National Computational Science Alliance under contract ACI-9619019 and by Department of Energy under the Accelerated Strategic Computing Initiative (ASCI) Academic Strategic Alliance Program (ASAP) Level 2, under subcontract No.W-7405-ENG-48 from Lawrence Livermore National Laboratories. This research was also supported in part by NSF cooperative agreement ACI-9619020 through computing resources provided by the National Partnership for Advanced Computational Infrastructure at the San Diego Supercomputer Center. We also acknowledge the use of the SGI Origin2000 at NCSA, and the Linux Cluster at ANL. We are grateful to Joseph Zambreno and Jayaprakash Pisharath for their valuable feedback.

Reference

[1] Daniel J. Eisenstein, Piet Hut, "Hop: A New Group Finding Algorithm for N-body Simulations," *Astrophysics. J.* 498, 137-142, 1998.

[2] University of Washington, "N-Body Shop," *http://www-hpcc.astro.washington.edu/tools/smooth.html.*

[3] J.L. Bentley, "Multidimensional Binary Search Trees Used for Associative Searching," *Communication of the ACM*, 18(9), September 1975.

[4] David W. Pfitzner, John K. Salmon, Thomas Sterling, "Halo World: Tools for Parallel Cluster Finding in Astrophysical N-body Simulations," *Data Mining and Knowledge Discovery*, 419-438, Volume 1, No. 4, 1997.

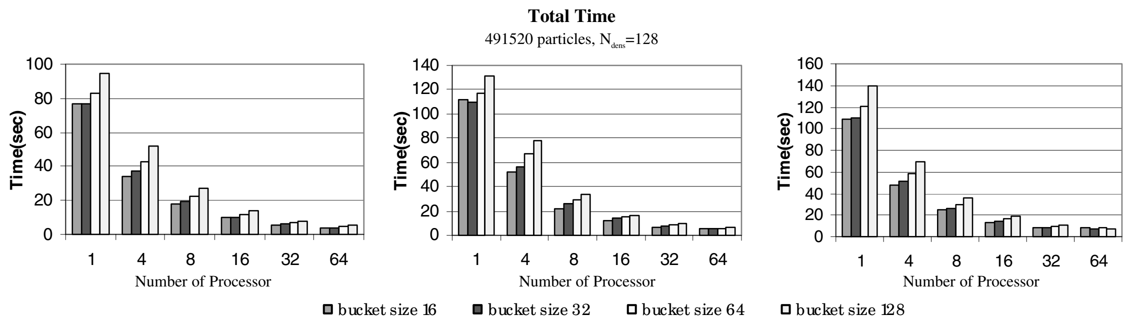
[5] Ying Zhao, George Karypis, "Criterion Functions for Document Clustering," *Technical Report, University of Minnesota,* tr# 01-40.

[6] K. Jain and R. C. Dubes, *Algorithms for Clustering Data.* Prentice Hall, 1998.

[7] B. King, "Step-wise clustering procedures," *Journal of the American Statistical Association*, 69:86-1-1, 1967.

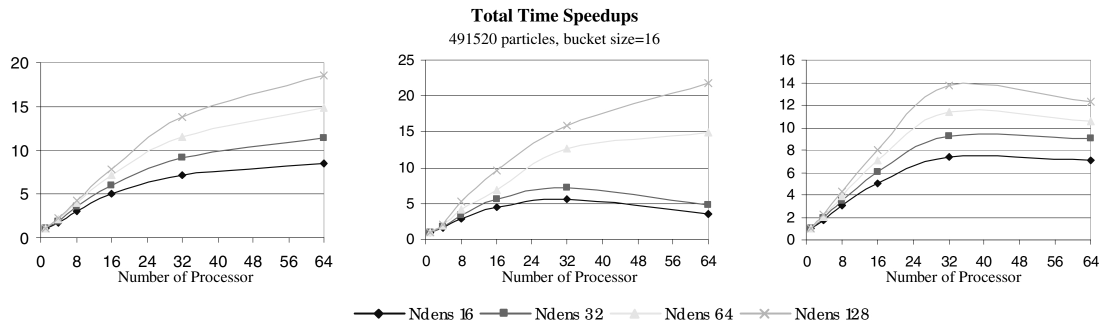
[8] Sudipto Guha, Rajeev Rostogi, and Kyuseok Shim, "CURE: an efficient clustering algorithm for categorical attributes," in *Proc. Of 1998 ACM-SIGMOD Int. Conf. on Management of Data*, 1998.

[9] Sudipto Guha, Rajeev Rostogi, and Kyuseok Shim. "ROCK: a robust clustering algorithm for categorical



(a) IBM SP2 (b) SGI Origin2000 (c) Linux Cluster

Fig. 13. Total time with varying bucket size on data set 1.



(a) IBM SP2 (b) SGI Origin2000 (c) Linux Cluster

Fig. 14. Total execution time speedups with varying N_{dens} on data set 1.

attributes," in *Proc. Of the 15th Int'l Conf. on Data Eng.*, 1999.

[10] G. Karypis, E.H. Han, and V. Kumar. "Chameleon: A hierarchical clustering algorithm using dynamic modeling," *IEEE Computer*, 32(8): 68-75, 1999.

[11] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. 5th Symp. Math. Statist, Prob.*, pages 281-297, 1967.

[12] P. Demspter, N. M. Laird and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the royal Statistical Society*, 39, 1977.

[13] K. Zahn, "Graph-theoretical methods for detecting and describing gestalt clusters," *IEEE Transactions on Computers*, (C-20): 68-86, 1971.

[14] R. Sibson, "SLINK: An optimally efficient algorithm for the single link cluster method," *Computer Journal*, 16:30-34, 1973.

[15] R. C. Prim, "Shortest connection networks and some generalizations," *Bell System Technical Journal*, 36:1389-1401, 1957.

[16] J. H. Ward Jr., "Hierarchical grouping to optimize an objective function," *Journal of the American Statistical Association,* 58:236-244, 1963.

[17] M. Davis, G. Efstathiou, C. S. Frenk, S. D. M. White, "The evolution of large-scale structure in a universe dominated by cold dark matter," *Astrophysical J.*, 292:371-394, 1985.

[18] J. M. Gelb and E. Bertschinger, "Cold dark matter 1: The formation of dark halos," *Astrophysical J.*, 436:467-490, 1994.

[19] George Forman, Bin Zhang, "Linear Speed-Up for a Parallel Non-Approximate Recasting of Center-based Clustering Algorithms, including K-Means, K-Harmonic Means, and EM," *ACM SIGKDD Workshop on Distributed and Parallel Knowledge Discovery*, KDD-2000, Boston, MA, August 20, 2000.

[20] G. Bryan, T. Abel, and M. Norman, "Achieving Extreme Resolution in Numerical Cosmology Using Adaptive Mesh Refinement: Resolving Primordial Star Formation," *SupuerComputing Conference*, Nov., 2001.

[21] M. Norman, J. Shalf, S. Levy, and G. Daues, "Diving Deep: Data-Management and Visualization Strategies for Adaptive Mesh Refinement Simulations," *Computing in Science and Engineering*, 1(4), pp. 36-47, Jul/Aug, 1999.

[22] Jianwei Li, Wei-keng Liao, Alok Choudhary, Valerie Taylor, "I/O Analysis and Optimization for an AMR Cosmology Application," in *Proceeding of Cluster 2002*, Sept. 2002.

[23] Message Passing Interface Forum, "MPI: A Message Passing Interface Standard," *http://www.mpi-forum.org/docs/docs.html*, June, 1995.

[24] Thomas H. Cormen, Charles E. Leiserson, Ponald L. Rivest, "Introduction to Algorithms," MIT press, 1990.