

# An Adaptive Parallel Hierarchical Clustering Algorithm

Zhaopeng Li, Kenli Li\*, Degui Xiao, and Lei Yang

School of Computer and Communication, Hunan University, Changsha, 410082, P.R. China  
lzpdc@163.com, LKL510@263.net, jt\_dgxiao@hnu.cn,  
leideyang@tom.com

**Abstract.** Clustering of data has numerous applications and has been studied extensively. It is very important in Bioinformatics and data mining. Though many parallel algorithms have been designed, most of algorithms use the CRCW-PRAM or CREW-PRAM models of computing. This paper proposed a parallel EREW deterministic algorithm for hierarchical clustering. Based on algorithms of complete graph and Euclidean minimum spanning tree, the proposed algorithms can cluster  $n$  objects with  $O(p)$  processors in  $O(n^2/p)$  time where  $1 \leq p \leq \frac{n}{\log n}$ . Performance comparisons show that our algorithm is the first algorithm that is both without memory conflicts and adaptive.

## 1 Introduction

Hierarchical clustering [2,3] techniques are widely applied in diversified areas such as gene categorization in biology, image processing, and network intrusion detection. Cluster analysis [1] is the process of classifying objects into subsets that have meaning in the context of a particular problem.

The basic principle behind hierarchical clustering is as follows: if there are  $n$  input points or data items, we start with  $n$  clusters where each cluster has a single point. From there on, the “closest” two clusters are identified. The two closest clusters are merged, resulting in a reduction in the number of remaining clusters is  $q$  where  $q$  is the target number of clusters and could be a part of the input.

The distance between two clusters can be defined in many ways. The commonly employed metric is the single link metric, as others did [2-4], and we also employ the single link metric as well.

Efficient sequential and parallel clustering algorithms have been studied extensively from researchers. By far the runtime of  $O(n^2)$  is the lowest to reach the goal of clustering  $n$  points under the single link metric [2,3]. Rasmussen and Willett [5] discuss the parallel hierarchical clustering using the single link metric and the minimum variance metric on a SIMD array processor. Although a significant constant factor speedup is achieved, their parallel algorithm can not decrease the  $O(n^2)$  time required by the serial implementation. Li and Fang [6] describe the parallel algorithms for hierarchical clustering using the single link metric on an  $n$ -node SIMD supercube

---

\* Corresponding author.

and an  $n$ -node butterfly respectively. They affirmed that their algorithms run in  $O(n \log n)$  time on the hypercube and  $O(n \log^2 n)$  on the butterfly. But in fact it was pointed out that the times required by their algorithms must be  $O(n^2)$  [2]. Olson has

presented  $O(n \log n)$ -time  $\frac{n}{\log n}$ -processor algorithms on the Concurrent-Read-

Concurrent-Write (CRCW) Parallel Random Access Machine (PRAM), butterfly, and tree models [2]. An  $O(n^2)$ -time  $n$ -processor SIMD shuffle-exchange network algorithm has also been given by Li [7]. E. Dahlhaus [3] develop an efficient parallel algorithm for single linkage clustering in  $O(\log n)$  time with  $O(n)$  processors on a Concurrent-Read-Exclusive-Write (CREW) PRAM, given that a Minimum Spanning Tree (MST) is known. More recently, Rajasekaran [4] has presented a SIMD algorithm that runs in  $O(\log n)$ -time using  $\frac{n^2}{\log n}$  CRCW PRAM processors, and a

Random algorithm that run in an expected  $O(\log^2 n)$  cycles on a  $1 \times n$  AROB.

However, as pointed out in literature [9,10,11], most of the parallel algorithms for hierarchical clustering presented so far have an obvious drawback: they are all impractical, and thus are of only theoretic importance. For example, if a large database has 10000 objects (this is possible in such cases as in bioinformatics and in intrusion detection), to perform the algorithms in [2] and [4], then at least 1000 and  $10^7$  processors which shared a common memory must be needed respectively. However, it is very difficult to build such a shared memory machine by the present techniques [9,10]. Another drawback of these algorithms resides in the parallel models they used. Since concurrent read or write are required in these algorithms, it is not applicable for the EREW (Exclusive-Read-Exclusive-Write) PRAM.

To overcome these two drawbacks, we propose an adaptive parallel hierarchical algorithm based on EREW. The contribution of our algorithm is as follows:

- Using  $p$  processors where  $p$  can be adjustable in the range from 1 to  $1 \leq p \leq \frac{n}{\log n}$ , our proposed algorithm run in  $O(n^2/p)$  time, according to the definition in [9,10,12], our algorithm is adaptive.
- An improvement from CRCW to EREW is made in this algorithm. Although any CRCW or CREW algorithm can be converted into EREW algorithm in a straightforward way [10], the time needed must increase by  $O(\log n)$  times.

The rest of this paper is organized as follows. Section 2 introduces our algorithm, in Section 3, the performance comparisons follow. Finally, some concluding remarks are given in Section 4 as well as some future research directions in this field.

## 2 The Proposed Algorithm

As we know[2,3,4], Hierarchical clustering may be represented by a dendrogram that can be easily constructed from a Euclidean minimum spanning tree of the  $n$  input

points [2]. So the main work for hierarchical clustering is to find EMST and identifying the connected components of a graph [2,3,4]. However, before the EMST can be constructed for  $n$  given points, one auxiliary thing has to be done is to obtain the distance of every pair of points and save them (i.e. to construct a completed graph for  $n$  points). Now we at first describe the parallel algorithms for this step.

Suppose that in SIMD-PRAM model we have  $p$  processors where  $1 \leq p \leq \frac{n}{\log n}$  and  $O(n^2)$  shared memory units, while each processor has  $O(1)$  local memory.

## 2.1 A Complete Graph Constructed in Parallel

Let the initial input points have  $m$  dimensions. At first we have to obtain all distances  $d_{ij}$  for all pairs of points  $V_i$  and  $V_j$  where  $1 \leq i, j \leq n$ . There are many methods to denote a completed graph  $G(V, E)$  [10]. Here, for the convenience of the following process, an adjacency matrix  $M$  is used. To construct an adjacency matrix for a given graph  $G$ , there are several parallel algorithms [11], but they are all based on SIMD-CRCW or SIMD-CREW models. Thus we use the elegant ideas [9,13] to design a parallel EREW algorithm for constructing an adjacency matrix  $M$  for a completed graph  $G$ .

**Algorithm 1.** Parallel algorithm for computing all distances  $d_{ij}$  of edges  $e_{ij}$ ,  $1 \leq i, j \leq n$

The  $n$  input points  $V_1(x_{11}, x_{12}, \dots, x_{1m}), V_2(x_{21}, x_{22}, \dots, x_{2m}), \dots, V_n(x_{n1}, x_{n2}, \dots, x_{nm})$  are given

```

begin
  for  $l = p$  to 1 step by  $-1$  do
    for all processors  $P_i$  where  $1 \leq i \leq l$  do
      for  $k = \left\lceil \frac{n}{p} \right\rceil (i-1 + p-l) + 1$  to  $\left\lceil \frac{n}{p} \right\rceil \times (i + p-l)$  do
        for  $j = \left\lceil \frac{n}{p} \right\rceil (i-1) + 1$  to  $\left\lceil \frac{n}{p} \right\rceil \times i$  do
          begin
            compute the Euclidean distance  $d_{kj}$  where  $d_{kj} = \sqrt{\sum_{s=1}^m (x_{ks} - x_{js})^2}$ 
            write  $d_{kj}$  to the shared memory
          end
        end
      end
    end
  end

```

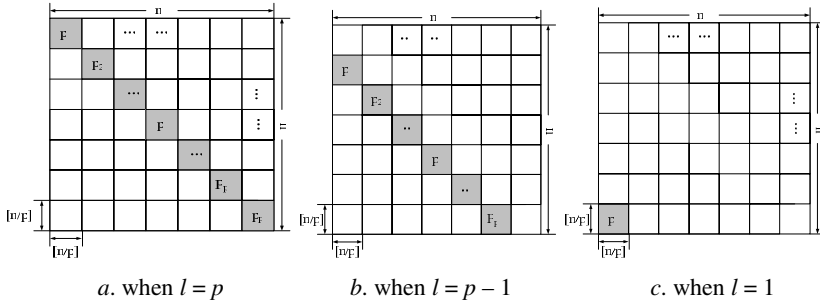
Here, as figure 2 shows, when  $p \mid n$ , algorithm 1 must perform well. However, it is trivial to process similarly in algorithm 1 when  $p \nmid n$  is not the case.

**Lemma 1.** Algorithm 1 can be efficiently executed on SIMD-PRAM without memory conflicts in  $O\left(\frac{n^2}{p}\right)$  time and  $O(n^2)$  work.

*Proof.* Assume that the time for computing the Euclidean distance for each PRAM processor is  $O(1)$  (in fact it must be  $O(m)$ , in this paper we let  $m$  be a constant), the conclusion can be shown from the figure 2. (i) if  $p|n$ . As the algorithm 1 shows, there are 3 for cycle to perform in order to execute algorithm. The time needed in the proposed parallel algorithm is  $p \times \frac{n}{p} \times \frac{n}{p} = \frac{n^2}{p}$ , and the total work (cost) is thus

equal to  $p \times \frac{n^2}{p} = n^2$ . When any processor has obtained the Euclidean distance  $d_{ij}$ , it

write it to the different location of the shared memory. Because every  $d_{ij}$  which different processors write are different from each other, there are no write conflicts among different processor. As figure 1 shows, in any time for different processors  $P_i$  and  $P_j$ , if  $i < j$ , then the subscripts of  $d$  computed by processor  $i$  will be totally different with that read by processor  $j$ , which means that the memory units processor  $i$  and  $j$  read are also different in any time. Thus there are no read conflicts among different processors, either. And algorithm 1 can be executed on EREW computation model. (ii) if  $p|n$  does not hold. In this case, except the final processor and in final cycle for  $l$ , the situations for other processors and other cycle are as same as that in case 1, and the time and work bound will not change.



**Fig. 1.** The data allocation in different processor as algorithm 1 executes

### 2.2 Generation of MST

Now we have known all distances  $d_{ij}$  from points  $V_i$  and  $V_j$ ,  $1 \leq i, j \leq n$ , which make up a symmetry adjoin matrix. Based on this matrix, the next task for clustering is to generate a MST of completed graph  $G$ . parallel algorithms for generating MST have been extensively researched in past twenty years, and there are many different parallel algorithms to get a MST from a known graph [12,14]. However, because of our objective is to cluster  $n$  input points (items), which require pruning some edges (section 2.3) after getting a MST, we use the parallel algorithms proposed by Akl, etc[12]. This parallel algorithm is an adaptive and cost-optimal parallel algorithm for MST, and is without memory conflicts.

**Algorithm 2.** The parallel algorithm for producing MST<sup>[12]</sup> Limited by the space, for more details on the parallel algorithms, one can refer to [10,12].

**Lemma 2.** Algorithm 2 can be execute in PRAM-EREW, and the total time and work for execute the algorithm 2 is  $(n^2/p)$ , and  $O(n^2)$ , respectively, if  $1 \leq p \leq \frac{n}{\log n}$ .

*Proof.* Because the number of processors in the proposed algorithm is  $p$  where  $1 \leq p \leq \frac{n}{\log n}$ , the conclusions are validated [12].

### 2.3 Pruning Edges and Finding Connected Components

Now we have obtained the MST of the  $n$  input points. The  $q$  clusters from the  $n$  input points are in need, so  $q - 1$  longest edges must be pruned from the MST  $T$ . if this is done, then the left  $q$  subtrees (components) are the  $q$  clusters of interest. Thus there is still two parts of work to do, one is pruning edges from  $T$ , and the other is finding the needed connected components or subtrees.

At first, we introduce the parallel pruning edges algorithm on EREW-PRAM.

**Algorithm 3.** Pruning edges algorithm

```

begin
  for  $k = 1$  to  $q - 1$  do
    for all processor  $P_i$  where  $1 \leq i \leq p$  do
      find the longest edges from the edges set of TREE[  $(i - 1) \lfloor \frac{n - 1}{p} \rfloor + 1$  ] to
      TREE[  $(i \lfloor \frac{n - 1}{p} \rfloor)$  ] do
        call Procedure MAXIMUM
        delete the longest edge
        call Procedure BROADCAST in [10,12]
    end
  Procedure MAXIMUM(  $A[1, \dots, p]$  )
  begin
    for all  $p_i$  where  $1 \leq i \leq \lfloor \frac{p}{2} \rfloor$  do
      for  $j = 1$  to  $\lceil \log n \rceil$  do
        if  $(i \bmod 2^{j-1}) = 0$  and  $2i + 2^{j-1} \leq p + 1$  then
          if  $A[2i - 1] < A[2i - 1 + 2^{j-1}]$  then  $A[2i - 1] = A[2i - 1 + 2^{j-1}]$  end if
        end if
      end
    end
  end

```

**Lemma 3.** Algorithm 3 can be executed on EREW-PRAM in

$$O\left(\frac{n}{p} + 2\log p\right) \times (q - 1) \text{ time.}$$

*Proof.* It is known that finding the maximum can be finished in linear time with the number of items. In our algorithm, each processor takes  $O([(n-1)/p])$  time to find the longest edge from its  $[(n-1)/p]$  edges. The time needed for running subprocedure BROADCAST [12] and MAXIMUM are both  $O(\log p)$ . To obtain  $q$  clusters from the  $n$  input points, the circle will have to be run for  $q - 1$  times. Thus the total time for

running the algorithm 3 is  $O\left(\frac{n}{p} + 2\log p\right) \times (q - 1)$ . Obviously, as we almost evenly divide the array TREE[ $n - 1$ ] into  $p$  blocks, and the subprocedure MAXIMUM is designed to perform on EREW, there is no memory conflicts in algorithm 3.

After pruning edges from the MST, the final step to cluster  $n$  objects is to decide which points are in the same cluster. In fact, there left  $q$  connectivity components or subtrees after performing pruning procedure. So the task of this step is finding the  $q$  connectivity components. Parallel connectivity algorithms had once been extensively researched in past years, and there are many such parallel algorithms [15,16]. For more details on the parallel algorithms for this problem, one can refer to [15,16].

However, none of these algorithms are of our need in the case that the graph has become a forest which is saved as an array TREE[ $n - 1$ ], either for their not having adaptivity, or for their time complexity. Therefore, we designed a new parallel connectivity algorithm to our need, which is both adaptive and without memory conflicts.

In our algorithm for finding the  $q$  connectivity components or clusters, an array  $D(n)$  is defined, which is used to store the number of the connectivity components, and at first it is initialized as  $D(i) = i$ . After the algorithm is performed, if node  $i$  and  $j$  are in the same cluster  $k$ , then  $D(i) = D(j) = k$ . for the convenience of describing the algorithm, assume that all edges of the forest are saved in array TREE[ $n - 1$ ] from TREE[0] to TREE[ $n - q - 1$ ]. If this condition is not satisfied, one can easily do this with  $O(n)$  time.

**Algorithm 4.** Algorithm for finding the  $q$  connected components of graph  $G$

```

begin
  for all processor  $P_i$  where  $1 \leq i \leq p$  do
    for  $u = \left\lfloor \frac{n}{p} \right\rfloor \times (i - 1) + 1$  to  $\left\lfloor \frac{n}{p} \right\rfloor \times i$  do
       $D(u) = u$ 
    for  $i = 0$  to  $n - q - 1$  do
       $v_l = \text{TREE}[i][1]$  and  $v_m = \text{TREE}[i][2]$ 
      if  $l > m$  then  $D(l) = D(m)$ 
      else  $D(m) = D(l)$ 
      end if
    call Procedure FIND
  end

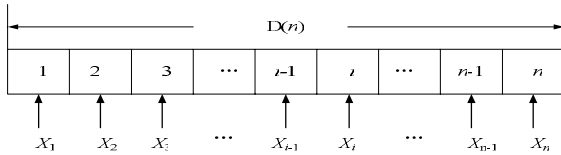
```

**Procedure FIND( $j, k$ )**

**for** all processor  $P_i$  where  $1 \leq i \leq p$  **do**

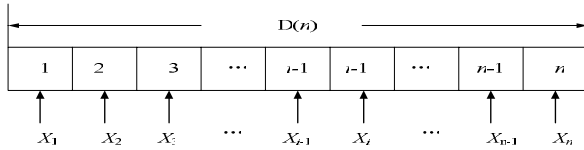
    find all  $D(t)$  from  $D[n/p \times i]$  to  $D[n/p \times i + n/p \times (i + 1) - 1]$  where  $D(t) = j$   
      $D(t) = k$

In algorithm 4, after the first cyclic is performed, the array  $D(n)$  is initialized, and the corresponding relation of array  $D(n)$  with the vertexes of the graph  $G(V, E)$  is depicted in Figure 2.



**Fig. 2.** The corresponding relation of the initialized array  $D(n)$  with all vertexes

While in the second cyclic, if two vertexes connected by the current input edge are  $x_{i-1}$  and  $x_i$  (ie.,  $TREE[k][1] = x_{i-1}$ , and  $TREE[k][2] = x_i$ ), then the value of  $i$ th element of array  $D$ ,  $D(i)$  will change into  $i - 1$ , as Figure 3 shows.



**Fig. 3.** If an edge  $(x_{i-1}, x_i)$  appears in the forest, the value of array  $D$  will be altered

If this is the case happened, the Procedure FIND is called to alter the values of all array elements whose value is being  $i$  to  $i - 1$ , for it is possible that one edge in the forest connecting  $v_i$  and  $v_j$  ( $j > i$ ) is processed before edge  $(x_{i-1}, x_i)$ . There are only  $n - q$  edges in forest, so after these codes are performed for  $n - q$  times, the label number stored in array  $D(n)$  is the number of cluster. While for certain cluster, all sequence number of array  $D(n)$  having same array element are the corresponding vertexes of this cluster.

**Lemma 4.** Algorithm 4 can be performed in  $O(n^2/p)$  time with  $p$  processors on PRAM-EREW where  $p \leq n$ .

*Proof.* The first cyclic can be finished in  $O(n/p)$  time. Since the subprocedure FIND takes at most  $O(n/p)$  time, the second cyclic will take  $O((n - q) \times n/p)$  time. Thus the total time to perform algorithm 4 is  $O(n/p) + O((n - q) \times n/p) = O(n^2/p)$ . As in the whole performing stage, all processor only read data from its memory units, and at the same time there are no different processors write into the same memory units. It is obvious that the algorithm can be performed on PRAM-EREW model.

## 2.4 The Proposed Parallel Hierarchical Clustering Algorithm

So far, we have finished the three steps of hierarchical clustering. Now combined the above four algorithms, one can obtain the parallel hierarchical cluster algorithm, which is based on the EREW model, and totally without memory conflicts.

**Algorithm 5.** Parallel algorithm for hierarchical cluster based on EREW

- Step 1. perform the parallel algorithm for computing the adjoin matrix.
- Step 2. perform the algorithm for producing the MST.
- Step 3. perform the parallel pruning edge algorithm.
- Step 4. perform the algorithm for recognizing connected components.

**Theorem 1.** Clustering  $n$  input points can be finished in  $O(n^2/p)$  time with  $p$  processors on EREW-PRAM model.

*Proof.* Following the lemmas 1 to 4, the total time needed to perform the proposed parallel cluster algorithm is  $T = n^2/p + n^2/p + ((\frac{n}{p} + 2\log p) \times (q - 1)) + n^2/p$ . Notice that in algorithm 2 and 4, the number of processors are limited in the range of  $1 \leq p \leq \frac{n}{\log n}$  and  $1 \leq p \leq n$  respectively. At the same time, the number of the needed clusters  $q$  satisfies that  $q \leq n$ . Thus the total time to perform the parallel clustering algorithm is bounded by  $O(n^2/p)$ . Since algorithms 1 to 4 are all based on SIMD-EREW model, it is obvious that the proposed parallel algorithm can be performed on EREW-PRAM. And it is also shown that in this algorithm, all memory conflicts which may happen among different processors are avoided.

## 3 Performance Comparisons

For the past years, hierarchical clustering has been extensively researched, and there are many parallel algorithms for it on different models, for example the algorithms in literature [2,3], [7-8,17]. These algorithms are mainly based on SIMD parallel computation models. Following the previous researches, the performance comparison will be described in terms of time-processor tradeoff, i.e., the cost of the parallel algorithm<sup>[9,10]</sup>. Rasmussen and Willett [5] were the first to discuss parallel implementations of clustering using the single link metric on a SIMD array processor. But they can not decrease the  $O(n^2)$  time required by the best serial implementation. Li and Fang [6] also presented parallel algorithms for hierarchical clustering using the single link metric on an  $n$ -node hypercube and an  $n$ -node butterfly. However, The time needed in their parallel algorithm is  $O(n^2)$ , too [6]. Later, an  $O(n^2)$ -time  $n$ -processor SIMD shuffle-exchange network algorithm has been given by Li [7]. Olson [2] showed that parallelism could accelerate to solve larger instances of this problem.

Their algorithm runs in  $O(n \log n)$  time by allowing  $\frac{n}{\log n}$  processors to currently

access  $O(n^2)$  units in shared memory. E. Dahlhaus’s efficient parallel algorithm [3] for single linkage clustering runs in  $O(\log n)$  time with  $O(n)$  processors on CREW-PRAM under condition that the a Minimum Spanning Tree of the  $n$  input points is given. Another PRAM algorithm recently presented by Rajasekaran [4] runs in  $O(\log n)$ -time using  $\frac{n^2}{\log n}$  CRCW processors, resulting to an  $O(n^2)$  computation cost.

The common characteristic of most of the above algorithms is that they are all based on the CRCW or CREW models, and thus there exist memory conflicts among different processors when they access the same memory units concurrently. Although It has been shown that every CREW or CRCW algorithms that require  $T(L)$  time using  $P(L)$  processors (where  $L$  denotes the size of input or output data) can be transformed into an EREW algorithm which requires time  $O(T(L) \log L)$  still using  $P(L)$  processors [10]. Since our proposed algorithm is designed for EREW model, hence. In practical, in according to the definition of cost for parallel computation [10], the cost of our algorithm is only  $1/(\log n)$  of the algorithms in [2,3,4]. Secondly, like the algorithms in [9,12,13], the number of processors in our algorithm can be adjusted from 1 to  $1/(\log n)$  in accordance to the scale of the problem and actual computation condition, while keeping the computation cost unchanged. It can provide the probability for clustering very large datasets. Therefore, according to the definition in [9,10,12], our algorithm is adaptive or scalable.

For the purpose of clarity, the comparisons of the mentioned parallel algorithms which are based on SIMD model for hierarchical clustering are depicted in Table 1. It is obvious that our parallel algorithm outtakes undoubtedly other parallel algorithms in the overall performance.

Recently, M. Dash etc [17] present a parallel algorithm for hierarchical clustering, and this algorithm has good theoretical and experimental performance as compared with the relevant algorithms, but that algorithm is not based on SIMD models and

**Table 1.** Comparisons of the parallel algorithms for Hierarchical Clustering

algorithms	Model	Time	Processor	Adaptability	Cost
Sequential [2]	sequential	$O(n^2)$	1	no	$O(n^2)$
Rasmussen and Willett[5]	CRCW	$O(n^2)$	$n / \log n$	no	$O(n^2)$
S.Rajasekaran[4]	CRCW	$O(\log n)$	$O(n^2 / \log n)$	no	$O(n^2)$
Li and Fang[6]	SIMD hypercube	$O(n \log n)$	$n$	no	$O(n^2 \log n)$
Li[7]	SIMD shuffle-exchang	$O(n^2)$	$n$	no	$O(n^3)$
Olson[2]	CRCW	$O(n \log n)$	$n / \log n$	no	$O(n^2)$
Dahlhaus[3]	CREW	$O(n)$	$n$	no	$O(n^2)$
Ours	EREW	$O(n^2/p)$	$p$	yes	$O(n^2)$

their metrics for distance are centroid method. Although their method is valuable on improving the practical performance of clustering algorithms, theoretically, it is not comparable with our proposed algorithm.

## 4 Conclusions

Based on the EREW-SIMD machine with shared memory model, a new parallel algorithm for hierarchical clustering is proposed. The proposed algorithm use  $p$ ,  $1 \leq p \leq \frac{n}{\log n}$ , processors to clustering  $n$  objects in  $O(n^2/p)$  time, resulting to  $O(n^2)$  computation work which is by far the lowest in serial way. Since the number of processors in our algorithm can be adjusted in the range from 1 to  $n/\log n$  according to the available computation resources and the scale of the problem to be processed, our algorithm is adaptive[9,10]. To our knowledge, it is the first parallel algorithm in the sense of both without memory conflicts and adaptive for the hierarchical clustering on PRAM model.

On the other hand, although the number of processors in the proposed algorithm can be adjusted, we still can't rely on this kind of algorithms to solve the large scale clustering instances, for it is not possible to construct such shared memory computers with a large number of computers. Although presently M. Dash etc's algorithm are also performed in shared memory multiprocessor system (MIMD) [17], the number of processors is not large enough to clustering large scale datasets, especially to cluster gene data clustering. Since the supercomputer systems are mainly based on cluster technology now, it is a worthwhile work to investigate on feasible parallel algorithms based on present main distributed memory system.

**Acknowledgments.** This research was supported by the National Natural Science Foundation of China under Grant No. 60603053 and the Key Project of Education Ministry of China under Grant No.105128.

## References

1. Han, J., Kamber, M.: Data Mining: Concepts and Techniques. Morgan Kaufmann Publishers, San Francisco (2000)
2. Olson, C.F.: Parallel Algorithms for Hierarchical Clustering. *Parallel Computing* 21, 1313–1325 (1995)
3. Dahlhaus, E.: Parallel Algorithms for Hierarchical Clustering and Applications to Split Decomposition and Parity Graph Recognition. *Journal of Algorithms* 36, 205–240 (2000)
4. Rajasekaran, S.: Efficient Parallel Hierarchical Clustering Algorithms. *IEEE transactions on parallel and distributed systems* 16(6), 497–502 (2005)
5. Rasmussen, E.M., Willett, P.: Efficiency of hierarchic agglomerative clustering using the ICL Distributed Array Processor. *Journal of Documentation* 45, 1–24 (1989)
6. Li, X., Fang, Z.: Parallel Clustering Algorithms. *Parallel Computing* 11, 275–290 (1989)
7. Li, X.: Parallel Algorithms for Hierarchical Clustering and Clustering Validity. *IEEE Trans. Pattern Analysis and Machine Intelligence* 12, 1088–1092 (1990)

8. Tsai, H.R., Horng, S.J., Lee, S.S., Tsai, S.S., Kao, T.W.: Parallel Hierarchical Clustering Algorithms on Processor Arrays with a Reconfigurable Bus System. *Pattern Recognition* 30, 801–815 (1997)
9. Akl, S.G.: Optimal parallel merging and sorting without memory conflicts. *IEEE Trans, Comput.* 36(11), 1367–1369 (1987)
10. chen, G.: Design and analysis of parallel algorithm. higher education press, Beijing (2002)
11. Datta, A., Soundaralakshmi, S.: Fast Parallel Algorithm for Distance Transform. *IEEE Transactions on Systems, Man, and Cybernetics* 33(5), 429–434 (2003)
12. Akl, S.G.: An adaptive and cost-optimal parallel algorithm for minimum spanning trees. *Computing* 3, 271–277 (1986)
13. Li, K.L., Li, Q.H., Li, R.F.: Optimal parallel algorithm for the knapsack problem without memory conflicts. *Journal of Computer Science and Technology* 19(6), 760–768 (2004)
14. Jun, M., Shaohan, M.: Efficient Parallel Algorithms for Some Graph Theory Problems. *J?of Comput?Sci?Technol* 8(4), 362–366 (1993)
15. Nath, D., Maheshwari, S.N.: Parallel algorithms for the connected components and minimal spanning tree problems. *Inf: Proc. Lett.* 14(1), 7–11 (1982)
16. Chong, K.W., Han, Y.J.: Concurrent Threads and Optimal Parallel MinimumSpanning Trees Algorithm. *Journal of the ACM* 48(2), 297–323 (2001)
17. Dash, M., Petrutiu, S., Scheuermann, P.: pPOP: Fast yet accurate parallel hierarchical clustering using partitioning. *Data & Knowledge Engineering* 61(3), 563–578 (2007)