A novel parallelization approach for hierarchical clustering

Z. Du *, F. Lin

BioInformatics Research Centre, Nanyang Technological University, Nanyang Avenue, Singapore 639798, Singapore

Received 25 August 2004; accepted 18 January 2005
Available online 10 May 2005

Abstract

Identification of groups of genes that manifest similar expression patterns is a key step in the analysis of gene expression data. Hierarchical clustering is developed for that purpose. A fundamental problem with the previous implementations of this clustering method is its limitation to handle large data sets within a reasonable time and memory resources. In this paper, we present a parallel approach for solving this problem. Implementation of the parallel algorithm is illustrated on data from high dimensional microarray experiments related to the gene expression in cancerous disease and Arabidopsis seedling growth. They show considerable reduction in computational time and inter-node communication overhead, especially for large data sets.

© 2005 Elsevier B.V. All rights reserved.

Keywords: Clustering; Parallelization; Gene expression

1. Introduction

Clustering gene expression data helps understanding gene functions and regulation networks, and assists in the diagnostic of disease conditions and effects of medicinal
treatments. It also has numerous applications in biology and other disciplines. Hierarchical clustering [3] is one of clustering algorithms and has been used extensively. In recent years, there was renewed interest in these clustering algorithms because of the rapid advancement of gene sequencing which allowed the development of new techniques to exploit this information.

Hierarchical clustering begins with an initial partition into singleton clusters by successive merging of clusters until all elements belong to the same cluster. All the distances between the items to be clustered have to be calculated in a distance matrix. At each step, joining the two closest items creates a node. Subsequent nodes are created by pairwise joining of items or nodes based on the distance between them, until all the nodes merge into a desired number of clusters. The completeness of hierarchical clustering is deterministic. A remarkable problem in implementation of the hierarchical clustering is its inability to handle large data sets within a reasonable time and memory resources.

Though most of the algorithms in the literature are sequential, parallel algorithms have also been proposed. Rasmusseen and Willett implemented parallel version of the SLINK algorithm [2,7], Prim’s minimal spanning tree algorithms [6] and Ward’s minimum variance methods were implemented on a SIMD array processor, Li and Fang [8] designed a parallel version of hierarchical clustering on an $n$-node hypercube and an $n$-node butterfly, Driscoll et al. [4] used a relaxed heap data structure to be applied to parallel clustering on a PRAM; Olson [1] also performed hierarchical clustering using several distance matrix based on PRAM. All these existing versions of parallelization were designed for SIMD and PRAM.

In this short communication, we present a parallelization approach for implementation of the hierarchical clustering on a cluster of compute nodes, using the message passing interface. These clusters have become popular in bioinformatics research labs. Our experimental results show that, for large data sets, we can achieve considerable reduction in computational time and inter-node communication overhead, although theoretically there does not exist a parallel algorithm for hierarchical clustering with linear speedup in distributed memory paradigm [1].

2. Methods

Based on a sequential version of hierarchical clustering program by de Hoon [5], we describe the parallel implementation of the clustering algorithm on a cluster of compute nodes.

The algorithm begins with a set of distinct points considered as a separate cluster. The two clusters are agglomerated according to some distance measures used for clustering. This is repeated until all of the points are agglomerated into one hierarchically constructed cluster.

Without losing the generality, we use the single link metric: the distance between two items $x$ and $y$ is the minimum of all pairwise distances between items contained in $x$ and $y$. No further distances need to be calculated once the distance matrix has been calculated. In each iteration, the algorithm involves calculation of the distances
between each pair of clusters. As the distance matrix is symmetric, with zeros on the main diagonal, only a triangular half (the lower triangle in our case) of the distance matrix is saved. Obviously, the complexity of computing time is $O(n^2)$ where $n$ is the number of clusters. We allocate $p$ processors for the time-independent tasks. Each processor stores and computes a distance matrix in space $O(n \times (n/p))$. The inter-cluster distances are easily computed with $O(n \times (n/p))$ of time complexity. For each iteration of loop, each processor finds the minimum of the nearest neighbor distances locally. The global minimum of the nearest clusters and its index are found and broadcasted to other processors by using MPI_Allreduce and MPI_Broadcast. Thus we can perform this procedure with a time complexity of $O(n^2/p)$ instead of $O(n^2)$.

The above steps need message passing among processors when merging and updating distance matrix of its own. Each processor updates the distances from the clusters that are relevant to the new clusters. The optimal case is with the time complexity of $O(n \log n)$, using $n/\log n$ processors.

Our implementation is on a multi-node cluster with 16 customized AlphaServer ES45 compute nodes, each node with 4 Alpha-EV68 1 GHz processors, 8 GB/s memory bandwidth and an interconnect PCI adapter capable of over 280 MB/s sustained bandwidth. At the heart of the cluster is the Quadrics 128-port interconnect switch chassis, delivering up to 500 MB/s per node, with 32 GB/s of cross-section bandwidth and MPI application latencies less than 5 ms.

3. Experimental results

The parallel program with MPI has been developed and tested with large microarray data sets. For uniprocessor performance, we used the sequential version as a baseline. We ran five trials of our parallel program on 4, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44 and 48 processors and reported average time.

To show the improvement of time scaling on very high dimensional data, we worked with a set of quantitative gene expression data from microarray experiments on cancer study [9]. The samples are labeled with their histological type, like CNS, renal, etc. and normal. Detection of the products of these genes can provide a way to detection and diagnosis of a cancer and find a way to eliminate the cancer cells, minimize toxicity to any normal cell. This large set of microarray data comprises 277 quantitative measurements of the expression of 7452 of genes in a diverse set of cancers in normal human tissues. The improvement of time scaling on different numbers of processors is presented by Fig. 1.

In this experiment with large number of experiments, it can be observed that the parallel version scales up as expected, up to 25 when running on 48 processors, and the elapsed time is reduced from 32 min on a single processor to 87 s on 48 processors with the same set of the genes.

To demonstrate the improvement of time scaling on different numbers of genes in large number of experiments, expression data on Arabidopsis seedling growth of different gene sizes are selected for clustering: 9217 genes expressed in 13 experiments and 11017 genes expressed in 14 experiments respectively, shown in Fig. 2.
Fig. 1. Elapsed time and speedup on different numbers of processors with tumor gene expression data.

Fig. 2. Elapsed time (ET) and speedup (SU) with different numbers of genes.
The proposed parallel clustering algorithm is for implementation on compute
clusters with a distributed memory architecture, and with large network bandwidth
and low inter-note messaging latency. The experimental results do show good perfor-
mance and exhibit high scalability on data size and experiment number. It provides a
fast and practicable approach for parallel clustering, for high dimensional gene
expression data.

The ISO C source code is available on request from the authors. An open source
implementation of MPI is available from http://www-unix.mcs.anl.gov/mpi.

References

heaps with applications to parallel computation, Communications of the ACM 31 (11) (1988) 1343–
1354.
(1957) 1389–1401.
Journal 16 (1973) 30–34.