



A parallel clustering algorithm on the star graph and its performance



Hamid Sarbazi-Azad^{a,b,*}, Hamid R. Zarandi^{b,c}, Mahdi Fazeli^d

^a Department of Computer Engineering, Sharif University of Technology, Tehran, Iran

^b School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran

^c Department of Computer Engineering and Information Technology, Amirkabir University of Technology (Tehran Polytechnic), Tehran, Iran

^d Department of Computer Engineering, Iran University of Science and Technology, Tehran, Iran

ARTICLE INFO

Article history:

Received 17 December 2009

Received in revised form 9 March 2013

Accepted 30 March 2013

Keywords:

Interconnection networks

Star graph

Clustering

Parallel algorithm

Complexity

ABSTRACT

In this paper, a parallel algorithm is presented for data clustering on a multicomputer with star topology. This algorithm is fast and requires a small amount of memory per processing element, which makes it even suitable for SIMD implementation. The proposed parallel algorithm completes in $O(K + S^2 - T^2)$ steps for a clustering problem of N data patterns with M features per pattern and K clusters where S and T are the minimum numbers such that $NM \leq S!$ and $KM \leq T!$, on the S -dimensional star graph.

© 2013 Elsevier Ltd. All rights reserved.

1. Introduction

Feature vector is a basic notion of pattern recognition. A feature vector v is a set of measurements (v_1, v_2, \dots, v_M) which map the important properties of a collection of data into a Euclidean space of dimension M [1]. A *clustering algorithm* partitions a set of feature vectors into clusters. It is a valuable tool in exploratory pattern analysis, and helps making hypotheses about the structure of data. It is important in syntactic pattern recognition, image segmentation, registration, and many other applications. There have been many methods proposed in the literature for clustering feature vectors [1–6].

One popular clustering technique is the *squared-error* algorithm. This clustering algorithm is as follows [7]. Let N be the number of patterns to be partitioned and M represent the number of features per pattern. Let $F[0 \dots N-1, 0 \dots M-1]$ be the feature matrix such that $F[i, j]$ denotes the value of the $(j+1)$ th feature in the $(i+1)$ th pattern. Let s_0, s_1, \dots, s_{K-1} be the K resulting clusters. Throughout the paper, we shall use terms “cluster k ” and s_k interchangeably. Each pattern belongs to exactly one of the clusters. Let $C[i]$ represent the cluster to which pattern i belongs. Thus, we can define s_k as

$$s_k = \{i | C[i] = k, 0 \leq i < N\}, \quad 0 \leq k < K. \quad (1)$$

Let $|s_k|$ be the cardinality or size of s_k . The center of cluster k is a vector of size M defined as

$$\text{center}[k, j] = \frac{1}{|s_k|} \sum_{i \in s_k} F[i, j], \quad 0 \leq j < M. \quad (2)$$

* Corresponding author at: Department of Computer Engineering, Sharif University of Technology, Tehran, Iran.
E-mail address: azad@ipm.ir (H. Sarbazi-Azad).

```

ALGORITHM One_Pass_Partitioning ( $N, M, K$ )
{
    // Step 1: Assign patterns to clusters
    FOR ALL  $i, 0 \leq i < N$ 
        // Find nearest cluster to pattern  $i$ 
         $NewCluster[i] \leftarrow q$  such that  $d2[i, q] = \min \{d2[i, k]\}$  for  $0 \leq k < K$ 

    // Step 2: Update clusters if necessary
    IF (FOR ALL  $i, 0 \leq i < N$ ,  $NewCluster[i] = C[i]$ ) THEN // Check terminate condition
        Terminate execution;
    ELSE
        // Set  $NewCluster$  as current cluster
        FOR ALL  $i, 0 \leq i < N$ 
             $C[i] := NewCluster[i]$ ;

    // Step 3: Update cluster centers
    FOR ALL  $i, 0 \leq i < K$ 
        FOR ALL  $j, 0 \leq j < M$ 
            Compute  $center[i, j]$  based on the new configuration;
}

```

Fig. 1. Algorithm *One_Pass_Partitioning* to find clusters with minimum squared-error patterns.

The squared distance d^2 between pattern i and cluster s_k is given by

$$d^2[i, k] = \sum_{j=0}^{M-1} (F[i, j] - center[k, j])^2. \quad (3)$$

The squared error for cluster k is defined as

$$E^2[k] = \sum_{i \in s_k} d^2[i, k] \quad (4)$$

and the squared error for the clustering is given by

$$Error = \sum_{i=0}^{K-1} E^2[i]. \quad (5)$$

When applying the squared-error clustering algorithm to a set of N patterns, the ultimate goal is to minimize the value of the squared error. To achieve this goal, the algorithm is realized with a different number of clusters (different values for K). To partition the patterns for each amount of clusters, we begin with an initial set of arbitrary cluster centers, and then we repeatedly perform the following two steps until no pattern changes its cluster: (1) we assign each pattern to the nearest cluster (one of the clusters whose center has the least Euclidean distance from the pattern); (2) having all patterns partitioned into different clusters, we compute the new center of each cluster.

Since different sets of initial cluster centers may result in different final configurations, the above approach does not guarantee achieving the best configuration for each amount of clusters.

Fig. 1 shows the pseudo code of one pass of a trivial algorithm to partition patterns into a given set of clusters. As shown in the figure, only one pass of the previously described algorithm has a time complexity of $O(NMK)$ on a uni-processor system. To achieve an acceptable clustering configuration (or finding an optimal K value), the above code must be executed several times. Thus, several works were conducted to parallelize and optimize a single pass of the clustering algorithm to expedite the overall process. For example, in [8] a clustering algorithm was developed for systolic arrays. A clustering algorithm was proposed for multiprocessors with orthogonally shared memories in [9]. An SIMD hypercube algorithm with a run time complexity of $O(K \log NM)$ and memory requirement of $O(K)$ for each processor on an SIMD hypercube with $N \times M$ processing elements was developed in [10]. Ranka and Sahni [7] decreased the run time complexity to $O(K + \log^2 K + \log NMK)$ on the same hypercube topology.

This paper¹ proposes a parallel algorithm for pattern clustering on the star graph with a run time of $O(K)$ and a memory usage of $O(1)$. The *star graph* was proposed in [12] as an attractive alternative to the hypercube topology for interconnecting processors in parallel computers. It has been extensively studied in different aspects and many algorithms have been designed for it including communication algorithms [13], Fourier transform [14], load balancing [15], and sorting [16]. Our

¹ An early version of this work was reported in [11].

algorithm combines several communication techniques in a novel way to perform pattern clustering on an NM -node star graph. This algorithm relies on window broadcasting communication at some stages during computation, as will be discussed later. It also uses a special kind of processor ordering introduced in [17] in order to assign the data to the PEs in the initialization phase. Due to careful scheduling used for the communication steps of the algorithm on a star graph, the algorithm can be realized on both SIMD and MIMD parallel machines.

The rest of the paper is organized as follows. Section 2 gives some preliminaries starting with a description of the star graph, and some useful definitions and tools that we shall later use to develop our parallel algorithm. Section 3 describes the proposed parallel algorithm in detail. Performance analysis of the proposed algorithm is reported in Section 4, and finally, Section 5 concludes the paper.

2. The star graph: properties and communication algorithms

This section gives some preliminaries necessary for developing our parallel algorithm.

2.1. The star graph

The n -star graph, denoted by S_n , has $n!$ vertices (or nodes) corresponding to the $n!$ permutations of n distinct symbols $1, 2, \dots, n$. A vertex corresponding to permutation $a_1 a_2 \dots a_{i-1} a_i a_{i+1} \dots a_n$ is connected to those vertices corresponding to permutations $a_i a_2 \dots a_{i-1} a_1 a_{i+1} \dots a_n$ for $2 \leq i \leq n$, (that is, those permutations resulting from interchanging the first symbol in the permutation $a_1 a_2 \dots a_{i-1} a_i a_{i+1} \dots a_n$ with any of the remaining $n-1$ symbols). The edge connecting the vertex associated with the permutation resulting from interchanging the first and the i -th symbol is called the i -th dimension edge or connection. Thus, we can define a function Γ to give the permutation address of the node connected to a given node $a_1 a_2 \dots a_n$ via the i -th connection as $\Gamma_i(a_1 a_2 \dots a_n) = a_i a_2 \dots a_{i-1} a_1 a_{i+1} \dots a_n$. In this way, every vertex is an endpoint of $n-1$ edges, corresponding to $n-1$ symbols that can be interchanged with the symbol in the first position of the associated permutation. This is shown in Fig. 2 for three different sizes of stars, the 2-star (S_2), 3-star (S_3) and 4-star (S_4). The degree and diameter of the n -star are of linear order, $O(n) = O(\log n! / \log n)$, while they are of higher order, $O(\log n!) = O(n \log n)$, for the equivalent hypercube (with the same number of nodes $N = n!$) [18].

2.2. Ordering nodes

We need an ordering for the nodes of the n -star and a function that maps this ordering to positive integers in order to develop our algorithm. The processor ordering defined in [12] is used where an ordering $<$ on the nodes of the n -star is defined as follows. We say $a_1 a_2 \dots a_i \dots a_n < b_1 b_2 \dots b_i \dots b_n$ if there exists an i , $1 \leq i \leq n$, such that $a_i > b_i$, and $a_j = b_j$ for all $j > i$. The binary relation $<$ is therefore a strict partial ordering being non-reflexive, transitive and asymmetric. Using this relation, we can order patterns in the graph. Let us now describe a function that maps the permutations ordered according to $<$ into the first $n!$ integers $1, 2, \dots, n!$. For each permutation $a_1 a_2 \dots a_i \dots a_n$, we define π_i for each a_i , $2 \leq i \leq n$, as

$$\pi_i = \left| a_i - i - \sum_{j=i+1}^n \langle a_i > a_j \rangle \right| (i-1)! \quad (6)$$

where

$$\langle a_i > a_j \rangle = \begin{cases} 1 & \text{if } a_i > a_j \\ 0 & \text{otherwise.} \end{cases} \quad (7)$$

Then the associated positive number for permutation $a_1 a_2 \dots a_i \dots a_n$ is given by

$$\Pi(a_1 a_2 \dots a_n) = 1 + \sum_{j=2}^n \pi_j. \quad (8)$$

For example, for $n = 4$, the permutations of $\{1, 2, 3, 4\}$ ordered according to $<$, and the associated positive numbers (Π) are given in Table 1. As shown in this table, $\Pi_{4321} = 1 + \pi_2 + \pi_3 + \pi_4 = 24$ (where $\pi_2 = 1$, $\pi_3 = 4$, $\pi_4 = 18$).

We shall interchangeably use $P_{a_1 a_2 \dots a_n}$ or $P_{\Pi(a_1 a_2 \dots a_n)}$ to indicate a processor node associated with the permutation address $a_1 a_2 \dots a_n$ in n -star [17].

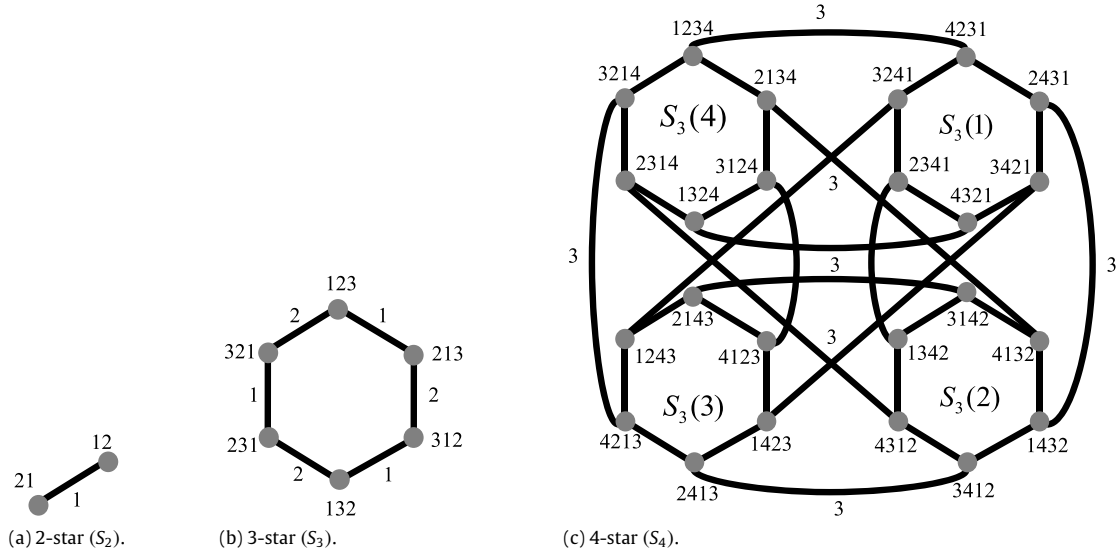
2.3. Routing and data communication in the star graph

This section first introduces some useful definitions, notations, and a routing algorithm that we shall use in the clustering process.

Definition 1. The notation $S_{n-1}(i)$, for $1 \leq i \leq n$, is used to indicate a sub-graph of S_n where every node address has i as its last symbol [13].

Table 1Permutation of the nodes in a 4-star ordered according to the ordering defined by \prec .

Permutation	Π	Permutation	Π	Permutation	Π	Permutation	Π
1234	(1)	1243	(7)	1342	(13)	2341	(19)
2134	(2)	2143	(8)	3142	(14)	3241	(20)
1324	(3)	1423	(9)	1432	(15)	2431	(21)
3124	(4)	4123	(10)	4132	(16)	4231	(22)
2314	(5)	2413	(11)	3412	(17)	3421	(23)
3214	(6)	4213	(12)	4312	(18)	4321	(24)

**Fig. 2.** Star graphs of 2, 3 and 4 dimensions.

A sub-star $S_{n-1}(i)$ is an $(n-1)$ -star defined on symbols $\{1, 2, \dots, n\} - \{i\}$. Thus, the n -star S_n can be decomposed into n sub- $(n-1)$ -stars, $S_{n-1}(i)$, for $1 \leq i \leq n$. For example, the 4-star S_4 in Fig. 2(c) contains four 3-stars, namely $S_3(1)$, $S_3(2)$, $S_3(3)$, and $S_3(4)$.

Definition 2. Let m_1 and m_2 be two distinct symbols from $\{1, 2, \dots, n\}$. We use notation $m_1 * m_2$ to represent a permutation of $\{1, 2, \dots, n\}$ whose first and last symbols are m_1 and m_2 , respectively, with $*$ representing any permutation of $n-2$ symbols in $\{1, 2, \dots, n\} - \{m_1, m_2\}$. Similarly, $m_1 *$ is a permutation of n symbols whose first symbol is m_1 , and $*m_2$ is a permutation of n symbols whose last symbol is m_2 [13].

Definition 3. Two or more nodes from distinct S_{k-1} 's are *corresponding* if they have the same index in their respective S_{k-1} 's according to the processor ordering scheme Π as in Eq. (8). For example the nodes with addresses 2341, 1342, 1243, 1234 are the corresponding nodes in S_4 .

In our parallel algorithm, a useful function called *Send* is used to transmit the contents of the nodes of $S_{k-1}(i)$ to the corresponding nodes of $S_{k-1}(j)$ in the host network is S_n . Therefore, the last $n-k$ symbols of the upper level S_k (in which the $S_{k-1}(i)$ and $S_{k-1}(j)$ are embedded) are the same. This function gets four values as inputs: i and j as the k -th symbols of two S_{k-1} 's, k as the dimension of the upper level sub graph in which $S_{k-1}(i)$ transmits its nodes' contents to the nodes in $S_{k-1}(j)$, and n as the dimension of the host network S_n . Notation $\delta_{k,n}$ represents the last $n-k$ similar symbols. This means that $\delta_{m,n}$ can be given as

$$\delta_{m,n} = \begin{cases} X_{m+1}X_m \dots X_n, & \text{if } m < n \\ \text{null}, & \text{if } m = n. \end{cases} \quad (9)$$

Fig. 3 shows the pseudo code of function *send*.

Rule 1. Every node value in a particular node of $S_{k-1}(i)$ is sent to its corresponding node in $S_{k-1}(j)$ using the *Send* function, if i and j are in a descending order in the symbol set (i.e. j is less than i and greater than the other remaining symbols).

Proof. Let $S = X_1X_2 \dots X_{k-1}X_k\delta_{k,n}$, $X_i \in \{1, 2, 3, \dots, n\}$, $1 \leq i \leq k$, be the source node in the particular S_{k-1} , and the *Send* function be used to transmit the node contents of $S_{k-1}(X_k)$ to $S_{k-1}(X_{k-1})$. The routing steps are as follows:

Step 1: $X_1X_2 \dots X_{k-1}X_k\delta_{k,n} \rightarrow X_kX_2 \dots X_{k-1}X_1\delta_{k,n}$

```

FUNCTION Send (i, j, k, n)
{
    FOR ALL  $s=i*\delta_{k,n}$  DO IN PARALLEL                                // Step 1
        Node  $P_s$  sends datum to its neighbor along connection k;
    FOR ALL  $w=i*\delta_{k,n}$ ,  $l \neq j$  DO IN PARALLEL                        // Step 2
        Node  $P_w$  send datum to its neighbor  $P_{j*\delta_{k,n}}$ ;
    FOR ALL  $v=j*\delta_{k,n}$ ,  $l \neq i$  DO IN PARALLEL                        // Step 3
        Node  $P_v$  sends datum along connection k;
    IF  $i=\text{minimum symbol } \{\text{Symbol Set} - \delta_{k,n}\}$  THEN
        FOR ALL u,  $1 \leq u \leq k-2$ 
            FOR ALL  $s=i*\delta_{k,n}$  DO IN PARALLEL
                Node  $P_s$  sends datum to its neighbor along connection u;
}

```

Fig. 3. Pseudo code of function *Send*.

Step 2: $X_k X_2 \dots X_{k-1} X_1 \delta_{k,n} \rightarrow X_{k-1} X_2 \dots X_k X_1 \delta_{k,n}$

Step 3: $X_{k-1} X_2 \dots X_k X_1 \delta_{k,n} \rightarrow X_1 X_2 \dots X_k X_{k-1} \delta_{k,n}$.

In this rule, for the sake of clarity, we suppose that $X_k > X_{k-1} > X_{k-2} \dots > X_1$. According to our processor ordering scheme node $X_1 X_2 \dots X_{k-1} X_k \delta_{k,n}$ has the least index in $S_{k-1}(X_k)$ and the node $X_1 X_2 \dots X_k X_{k-1} \delta_{k,n}$ also has the least index in $S_{k-1}(X_{k-1})$; therefore, the node $X_1 X_2 \dots X_k X_{k-1} \delta_{k,n}$ which is selected as a destination node is the *corresponding* node of the source node. \square

Rule 2. Two consecutive neighboring nodes v_1 and v_2 in $S_k(\alpha)$ send data to consecutive neighboring nodes w_1 and w_2 in $S_k(\beta)$, if α and β are in the descending order in the symbol set.

Proof. Suppose that node $v_1 = X_1 X_2 \dots \varphi \dots X_{k-1} \alpha \delta_{k,n}$ and node $v_2 = Y_1 Y_2 \dots \varphi \dots Y_{k-1} \alpha \delta_{k,n}$ in $S_{k-1}(\alpha)$ are two consecutive neighbors, and φ and α are in the descending order in the symbol set. After sending the contents of the nodes of $S_{k-1}(\alpha)$ to the nodes of $S_{k-1}(\varphi)$, we have the following steps for sending data from node $X_1 X_2 \dots \varphi \dots X_{k-1} \alpha \delta_{k,n}$:

Step 1: $X_1 X_2 \dots \varphi \dots X_{k-1} \alpha \delta_{k,n} \rightarrow \alpha X_2 \dots \varphi \dots X_{k-1} X_1 \delta_{k,n}$

Step 2: $\alpha X_2 \dots \varphi \dots X_{k-1} X_1 \delta_{k,n} \rightarrow \varphi X_2 \dots \alpha \dots X_{k-1} X_1 \delta_{k,n}$

Step 3: $\varphi X_2 \dots \alpha \dots X_{k-1} X_1 \delta_{k,n} \rightarrow X_1 X_2 \dots \alpha \dots X_{k-1} \varphi \delta_{k,n}$

and the following steps for sending data from node $Y_1 Y_2 \dots \varphi \dots Y_{k-1} \alpha \delta_{k,n}$:

Step 1: $Y_1 Y_2 \dots \varphi \dots Y_{k-1} \alpha \delta_{k,n} \rightarrow \alpha Y_2 \dots \varphi \dots Y_{k-1} Y_1 \delta_{k,n}$

Step 2: $\alpha Y_2 \dots \varphi \dots Y_{k-1} Y_1 \delta_{k,n} \rightarrow \varphi Y_2 \dots \alpha \dots Y_{k-1} Y_1 \delta_{k,n}$

Step 3: $\varphi Y_2 \dots \alpha \dots Y_{k-1} Y_1 \delta_{k,n} \rightarrow Y_1 Y_2 \dots \alpha \dots Y_{k-1} \varphi \delta_{k,n}$.

The nodes $w_1 = X_1 X_2 \dots \alpha \dots X_{k-1} \varphi \delta_{k,n}$ and $w_2 = Y_1 Y_2 \dots \alpha \dots Y_{k-1} \varphi \delta_{k,n}$ in $S_{k-1}(\varphi)$ are also consecutive neighboring nodes because exchanging symbols φ and α does not affect the ordering of the nodes. \square

Rule 3. In transmission from nodes of $S_{k-1}(\alpha)$ to nodes of $S_{k-1}(\beta)$, where α is the minimum symbol in the corresponding symbol set and β is the greatest one, $k-2$ exchange steps are required within $S_{k-1}(\alpha)$ to send data to the corresponding nodes in $S_{k-1}(\beta)$.

Proof. If the content of node $X_2 X_3 \dots X_k X_1 \delta_{k,n}$ is transmitted by function *send* to $S_{k-1}(X_k)$, where $X_k > X_{k-1} > \dots > X_1$, the following steps are performed in the first phase of the *send* function:

Step 1: $X_2 X_3 \dots X_k X_1 \delta_{k,n} \rightarrow X_1 X_3 \dots X_k X_2 \delta_{k,n}$

Step 2: $X_1 X_3 \dots X_k X_2 \delta_{k,n} \rightarrow X_k X_3 \dots X_1 X_2 \delta_{k,n}$

Step 3: $X_k X_3 \dots X_1 X_2 \delta_{k,n} \rightarrow X_2 X_3 \dots X_1 X_k \delta_{k,n}$.

It is clear that $X_2 \dots X_k X_1 \delta_{k,n}$ has the least index among other nodes in $S_{k-1}(X_1)$. Thus, in a correct transmission, the contents of this node should be transmitted to a node in $S_{k-1}(X_k)$ that has the least index (according to the processor ordering), but

use of the proposed algorithm does not accomplish this task in the first 3 steps. To do so, $k - 2$ exchange steps are required in $S_{k-1}(X_k)$ as follows:

$$\left. \begin{array}{l} \text{Step 1: } X_2 X_3 \dots X_1 X_k \delta_{k,n} \rightarrow X_3 X_2 \dots X_1 X_k \delta_{k,n} \\ \text{Step 2: } X_3 X_2 \dots X_1 X_k \delta_{k,n} \rightarrow X_4 X_2 X_3 \dots X_1 X_k \delta_{k,n} \\ \vdots \\ \text{Step } k-2: X_{k-1} X_2 X_3 \dots X_1 X_k \delta_{k,n} \rightarrow X_1 X_2 X_3 \dots X_{k-1} X_k \delta_{k,n} \end{array} \right\} \quad k-2 \text{ steps are required.}$$

From Rules 1–3, it can be concluded that each node in $S_{k-1}(\alpha)$ sends data to its corresponding node in $S_{k-1}(\beta)$ by the *send* function, if α and β are in the descending order, except when α is the minimum and β is the greatest symbol in the corresponding symbol set. The following transmission sequence shows a correct order of transmission:

$$S_{k-1}(X_k) \rightarrow S_{k-1}(X_{k-1}) \rightarrow \dots \rightarrow S_{k-1}(X_1) \rightarrow S_{k-1}(X_k) \quad \text{where } X_k > X_{k-1} > X_{k-2} > \dots > X_1. \quad \square$$

3. The parallel algorithm

The parallel algorithm consists of three main phases: *Initialization Phase*, *Cluster Finding Phase*, and *Center Update Phase*. The number of patterns in this algorithm, N , the number of clusters, K , and the number of features, M , should satisfy conditions $NM = S!$, $KM = T!$, and $M = R!$. If the number of patterns, clusters or features are not in a factorial manner, one can add enough dummy entries so that the above conditions are satisfied and the clustering results are not affected [7].

3.1. The initialization phase

During this phase, two index numbers are associated with each PE according to the mentioned node ordering scheme. The first index shows the order of PE in the host network S_S and the second one is the order of PE in the corresponding S_T . Then patterns are associated with different S_R 's in such a way that the i -th feature of each pattern resides on a PE whose index number satisfies condition $\text{index} \stackrel{M}{=} i$. Then, the first S_T in S_S is considered as the master cluster window (the choice of initial cluster centers in the master cluster window is arbitrary), and its contents are copied into all other S_T 's, so that the current cluster center selection is reported to all other cluster windows. Register R_1 of each node is used to store the squared distance of node to its cluster. Register R_2 is temporary and register R_3 represents the value of squared node distance to the current cluster. Registers F_1 and C_1 are used to store feature values and their cluster centers.

3.2. The cluster finding phase

The aim of this phase is to compute the distance $d^2(i, k)$ of the i -th pattern in each cluster window (i.e., the k -th cluster) from the current selection of cluster centers, and to choose the minimum distance to all cluster centers. We then assign this cluster to the pattern according to the selected choice.

First, distances between the features and current centers available in each node of a S_R are computed in a parallel fashion among all S_R 's in the network as $(F_1 - C_1)^2$.

Then by using function *Group Accumulate* [17], the value of $d^2(i, k)$ which represents the distance between the i -th pattern and the current center is calculated and compared to its old value; the smaller one is selected as the cluster to where this pattern belongs.

In the second step, the values of the S_R 's present in all the S_{R+1} 's available in the S_S are rotated once via the *send* function in parallel, as previously described. These steps will be repeated $R + 1$ times, until all S_R 's present in all S_{R+1} 's get each other's data.

The next step would be to rotate the data values of S_{R+1} 's in all S_{R+2} 's in parallel once, and repeat the first and second steps $R + 2$ times. The addition of the levels of sub-graphs and their rotation continues until S_{T+1} is reached; in other words, we reach one level higher than the cluster windows (S_T 's).

By the end of the *Cluster Finding Phase*, all the patterns have been assigned their cluster membership in the corresponding higher order node (according to PE ordering). These steps are shown in the pseudo code in Fig. 4.

In Fig. 4, $\delta(K)$ is a function that selects the maximum symbol among symbol set $\{X_1, X_2, \dots, X_{K-1}\}$ where the chosen symbol is less than X_K ; if this is not possible the maximum value in the symbol set is selected. Thus, $\delta(K)$ can be written as

$$\delta(K) = \begin{cases} \text{Max}\{X_1, X_2, \dots, X_{k-1} \mid X_i < X_k\}, & \text{if it exists} \\ \text{Max}(X_1, X_2, \dots, X_{k-1}), & \text{otherwise.} \end{cases} \quad (10)$$

In this figure, procedure *Split* divides a set, say $IJ = \{X_1, X_2, \dots, X_I\}$ into two subsets $I = \{X_1, X_2, \dots, X_{I/2}\}$ and $J = \{X_{I/2+1}, X_{I/2+2}, \dots, X_I\}$, where the division operator “/” means an integer division which results in an integer. Now, in the final phase, using these procedures the contents of registers R_1 in all processors are accumulated in register $P_{X_1, X_2, \dots, X_R \delta_{R,T}}(R_1)$, where $X_i > X_{i-1}$ [17].

```

FUNCTION Cluster_Finding (K)
{
1.   IF  $K \neq R$  THEN
2.       FOR ALL  $i, 1 \leq i \leq k$ 
3.           FOR ALL  $S_{K-1}$  IN  $S_K$  DO IN PARALLEL
4.                $Send(X_k, \delta(K), K, R)$  //  $X_k$  represent the  $k^{th}$  symbol of a particular node;
5.           Cluster_Finding ( $K-1$ );
6.   ELSE
7.       Compute_Distance ();
}

```

(a) Cluster finding function.

```

FUNCTION Compute_Distance ()
{
1.   FOR ALL  $S_R$  IN  $S_S$  DO IN PARALLEL
2.       FOR ALL nodes IN  $S_R$  DO IN PARALLEL
3.            $R_1 \leftarrow (F_1 - C_1)^2$ ;
4.       FOR ALL  $i \leftarrow R$  DOWN TO 2
5.            $IJ \leftarrow \{X_1, X_2, \dots, X_i\}$  where  $X_i > X_{i-1}$ 
6.       WHILE  $IJ \neq \{X_i\}$  DO
7.           Split ( $IJ, I, J$ );
8.           Group_Accumulate ( $I, J, i, R$ );
9.            $IJ \leftarrow J \cup \{X_i\}$ ;
10.       $P_{X_1, X_2, \dots, X_R \delta_{R,T}}(R_1) \leftarrow P_{X_1, X_2, \dots, X_R \delta_{R,T}}(R_2) + P_{X_1, X_2, \dots, X_R \delta_{R,T}}(R_1)$ ;
11.      FOR ALL PE's whose index MOD  $R! = 0$  DO IN PARALLEL
12.          IF  $R_1 < R_3$  THEN
13.               $R_3 \leftarrow R_1$ ;  $C_1 \leftarrow C_{Current}$ ;
}

```

(b) Compute Distance algorithm.

```

FUNCTION Group_Accumulate ( $I, J, m, n$ )
{
1.   FOR ALL  $k, 1 < k < m/2$  DO IN PARALLEL
2.       Accumulate ( $i_k, j_k, m, n$ );
}

```

(c) Group Accumulate function.

```

FUNCTION Accumulate ( $i, j, m, n$ )
{
1.   FOR ALL  $s = X_1 X_2 \dots X_{m-1} i \delta_{m,n}, (X_l \in \{1, 2, \dots, n\} - \{j\})$  DO IN PARALLEL
2.        $P_{iX_2 \dots X_{m-1} X_1 \delta_{m,n}}(R_2) \leftarrow P_s(R_1)$ 
3.   FOR ALL  $w = iX_1 X_2 \dots X_{m-2} k \delta_{m,n}, (X_l, k \in \{1, 2, \dots, n\} - \{i, j\})$  DO IN PARALLEL
4.        $P_{jX_2 \dots X_{l-1} X_{l+1} \dots X_{m-1} k \delta_{m,n}}(R_2) \leftarrow P_w(R_2)$ 
5.   FOR ALL  $v = jX_1 X_2 \dots X_{m-2} k \delta_{m,n}, (X_l, k \in \{1, 2, \dots, n\} - \{i, j\})$  DO IN PARALLEL
6.        $P_{kX_1 X_2 \dots X_{m-2} j \delta_{m,n}}(R_2) \leftarrow P_v(R_1)$ 
7.        $P_{kX_1 X_2 \dots X_{m-2} j \delta_{m,n}}(R_1) = P_{kX_1 X_2 \dots X_{m-2} j \delta_{m,n}}(R_2) + P_{kX_1 X_2 \dots X_{m-2} j \delta_{m,n}}(R_1)$ 
}

```

(d) Accumulate function.

Fig. 4. Functions in Cluster Finding phase.

3.3. The center update phase

As mentioned before, all cluster windows (S_T 's) have been indexed such that every node contains a variable $T = \lfloor \frac{\text{Index}}{R!} \rfloor$ that shows the cluster center data each S_R in a cluster window is responsible for. This value is a pre-computed constant for each S_R (which would contain a feature vector). This phase has two steps: *Broadcast Cluster Center* and *Cluster Center Update*.

3.3.1. Broadcast cluster center

Here, all S_R 's in S_S broadcast their cluster numbers computed in the previous phase and stored in their highest indexed node.

As mentioned earlier, the center number of patterns is stored in a node with the highest index within the S_R 's. Let $X_1X_2 \dots X_L\delta_{R,T}$ be the node with the highest index in the corresponding S_R . The following optimal broadcasting algorithm [19] can broadcast the content of this node to the PEs in $S_{R-1}(X_L)$.

Definition 4. Let the $\Pi\Delta$ -decomposition of a star graph S_n be defined as follows. Let Π_i , $1 \leq i \leq n$, be the graph induced by the set of vertices of the form $X_1X_2 \dots X_{n-1}i$ and let Δ_j , $1 \leq j \leq n$, be the graph induced by the set of vertices of the form $jX_2 \dots X_n$. The node $X_1X_2 \dots X_L\delta_{R,T}$ whose data shall be broadcast belongs to both sub-graphs Π_{X_L} and Δ_{X_1} .

Now, three actions must be taken in this step as:

- (1) *Merging*: broadcast the information from Δ_{X_1} to every other Π_{X_L} , $X_L > X_1$.
- (2) *Placement*: send the information from every Δ_{X_i} to every Π_{X_i} , for all $X_i \neq X_1$.
- (3) *Recursion*: broadcast within each Π_{X_i} .

This step has a run time of $O(n + \sum_{k=3}^{n-1} \lceil \log k \rceil)$.

3.3.2. Cluster center update

In this phase, all S_R 's of the S_{R+1} exchange their values $R + 1$ times. In each rotation step, the contents of nodes in S_R 's, which include cluster number and feature value are transmitted to their corresponding node in the next S_R . In each node of S_R 's, if the cluster number T is equal to the cluster number it receives (from the previous window S_R), the PE adds its feature value to the feature value it gets, otherwise it does nothing.

Next, the dimension of the last step is increased once, in fact S_{R+1} 's exchange data $R + 2$ times inside the corresponding S_{R+2} . In each exchange operation, the above steps are repeated again until S_{T+1} is reached.

Through the last step, all S_T 's of a S_{T+1} exchange their nodes' contents with their corresponding S_T 's, $T + 1$ times. Since corresponding S_R 's in two different S_T 's (S_R 's with similar values of T) contain similar cluster center information, the nodes of each S_T just add their former contents to the newly received ones; there is no need for any comparison or similar operation.

The last step is repeated until we reach S_5 . There will be $\frac{S(S+1)}{2} - \frac{T(T+1)}{2} + K$ addition and send operations. The pseudo code of this phase is shown in Fig. 5.

4. Performance analysis

In this section, we evaluate the performance of the proposed clustering algorithm. To this end, we suppose T_{link} and T_{cpu} being the time needed for communication over a link and calculating an expression inside a PE, respectively.

4.1. The initialization phase

The first step in the proposed algorithm is to scatter data among PEs. This step is done once during the algorithm. Lets T_{init} be the time needed for this step.

4.2. Cluster finding phase

Fig. 6 shows the required time for each of the functions for finding clusters in the proposed algorithms based on Fig. 4. In each part of this figure, the time consumed at each step (or line of the algorithm) is specified. Next, the accumulated execution time of each algorithm is shown at the bottom of each table.

Based on these estimated times, it is easy to find the overall execution time of *Cluster_Finding* algorithm.

4.3. The center update phase

Moreover, Fig. 7 portrays execution time of each function for finding clusters in the proposed algorithm based on Fig. 5. For each algorithm in *Center_Update* phase, the execution time is estimated and shown in Fig. 7. Therefore, the total execution time of the *Cluster_Center_Update* algorithm can be easily obtained.

Since the proposed algorithm is synchronous and there is no network contention, the execution time of the algorithm can be analytically calculated based on the figures above. In order to better understand the effect of different parameters on the overall performance, MATLAB tool version 2010a (7.10.0) is used to investigate the effect of each parameter on the total execution time. Without loss of generality, we have fixed the parameters shown in Table 2. It is noteworthy that different values of these parameters may change the calculated latency but will not change the final conclusion. As we want to see the effects of the number of patterns, features and clusters on the total execution time, other parameters listed in Table 2 are fixed to some constant values, i.e. time complexity of $O(1)$.

Figs. 8–10 show the effects of different parameters on the total execution time of the proposed algorithm. In Fig. 8, it is shown that the number of patterns (N) has a logarithmic impact on the total execution time. It is because in our algorithm, parameter N contributes in parameter S with an inverse factorial relation. Fig. 9 also shows the effect of the number of

```

FUNCTION Center_Compute (m)
{
1.   IF  $m \neq R+1$  THEN
2.       Center_Compute ( $m-1$ );
3.   ELSE
4.       {
5.           FOR ALL  $i, 1 \leq i \leq m$ 
6.           {
7.               FOR ALL  $S_{m-1}$  IN  $S_m$  DO IN PARALLEL
8.                   Send ( $X_m, \delta(m), m, R$ );
9.               FOR ALL nodes IN  $S_S$  DO IN PARALLEL
10.                  IF Current Cluster Number  $\leftarrow T$  (where  $T = \text{Index}/R!$ ) THEN
11.                       $F_1 \leftarrow F_1 + F_{\text{Current}}$ ;
12.                  /* it means the current received feature belongs to a pattern which has the
13.                     same cluster number as the current feature */
14.              }
15.          }
}

```

(a) Center computer function.

```

FUNCTION Accumulate_Center (n)
{
1.   IF  $n \neq T+1$  THEN
2.       Accumulate_Center ( $n-1$ )
3.   ELSE
4.       {
5.           FOR ALL  $S_{n-1}$  IN  $S_n$  DO IN PARALLEL
6.               Send ( $X_m, \delta(m), m, R$ );
7.           FOR ALL nodes IN  $S_S$  DO IN PARALLEL
8.                $F_1 \leftarrow F_1 + F_{\text{Current}}$ ;
9.       }
}

```

(b) Algorithm to accumulate center.

```

FUNCTION Cluster_Center_Update (S)
{
1.   FOR ALL  $S_R$  IN  $S_S$  DO IN PARALLEL
2.       Broadcast the highest order node to the other nodes in corresponding  $S_R$ 
3.   FOR ALL  $S_T$  's IN  $S_S$  DO IN PARALLEL
4.       Center_Compute ( $T$ );
5.   Accumulate_Center ( $S$ );
}

```

(c) Algorithm to update center in clusters.

Fig. 5. Pseudo code for Cluster Center Update step.**Table 2**
Parameters used in the simulation tool.

Parameter	Latency (cycles)
T_{link}	1
T_{cpu}	1
T_{split}	10
T_{union}	10

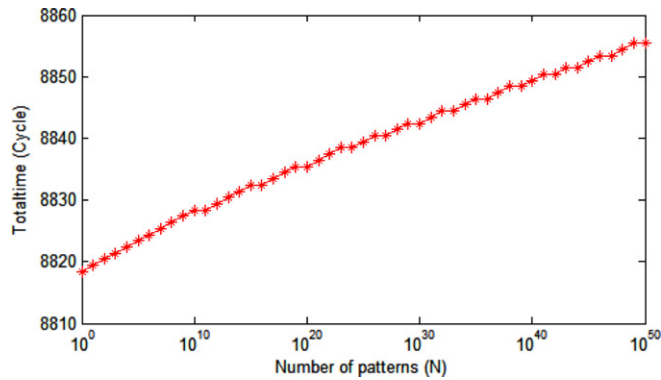
clusters (K) on total execution time. This figure shows that parameter K has an exponential effect on total execution time. Finally, Fig. 10 shows the effect of the number of features (M) on total execution time with a logarithmic contribution. As can

a Send () function Step1: T_{link} Step2: T_{link} Step3: T_{link} Step4: Worst case: $(K - 2) \times T_{link}$ other cases: 0 $T_{send} = 3 \times T_{link} + \frac{1}{K} \times (K - 2) \times T_{link}$	b Cluster_Finding () function If $K < R$ then return $K \times T_{send} + T_{ClusterFinding}(K - 1)$ Else return $T_{ComputeDistance}$ $T_{ClusterFinding}(K) = T_{send} \times \frac{K + R}{2} \times (K - R + 1) + T_{ComputeDistance}$
d Group_Accumulate (m) function Line 1: $\frac{m}{2}$ times Line 2: $T_{Accumulate}$ $T_{GroupAccumulate} = \frac{m}{2} \times T_{Accumulate}$	c Compute_Distance () function Lines 2-3: T_{cpu} Line 4: $(R - 1) \text{ times}$ Line 5: $O(1)$ Line 6: $\log_2(R)$ times Line 7: T_{split} Line 8: $T_{GroupAccumulate}(\frac{R}{2})$ Line 9: T_{union} Line 10: $T_{cpu} \times O(1)$ Lines 11-13: $T_{ComputeDistance} = T_{cpu} + (R - 1) \times \left\{ O(1) + \log_2(R) \times \left(T_{split} + T_{GroupAccumulate}(\frac{R}{2}) + T_{union} \right) + T_{cpu} \right\} + O(1)$
e Accumulate () function Lines 1-2: T_{link} Lines 3-4: T_{link} Lines 5-6: T_{link} Line 7: T_{cpu} $T_{Accumulate} = 3 \times T_{link} + T_{cpu}$	

Fig. 6. Timing analysis of Cluster Finding phase.

a Center_Compute()function Line 1: $(m - R) \text{ times}$ Line 2: $O(1)$ Line 5: $(m) \text{ times}$ Lines 7-8: T_{send} Lines 9-13: T_{spu} $T_{CenterCompute}(m) = (m - R + 2) \times O(1) + m \times (T_{send} + T_{cpu})$	b Accumulate_Center()function Line 1: $(n - T) \text{ times}$ Line 2: $O(1)$ Lines 5-6: T_{send} Lines 7-8: T_{spu} $T_{AccumulateCenter}(n) = (n - T) \times O(1) + T_{send} + T_{cpu}$
c Cluster_Center_Update()function Lines 1-2: $T_{Broadcast}$ Lines 3-4: $T_{Center_Compute}$ Line 5: $T_{Accumulate_Center}$ $T_{ClusterCenterUpdate} = T_{Broadcast} + T_{CenterCompute} + T_{AccumulateCenter}$	

Fig. 7. Timing analysis of Cluster Center Update phase.

Fig. 8. Total time consumption of the algorithm vs. number of patterns N (#features = 200, #clusters = 64).

be seen in the figure, for some consecutive values of M , the total execution time does not change which is due to rounded inverse-factorial values.

Fig. 11 shows the effect of the number of clusters on the performance of proposed algorithm compared to the algorithm presented in [7] for the same sized hypercube (let it be called the hypercube algorithm). Since the hypercube algorithm can be used only for situations where the number of clusters is a power of 2, we have compared the two algorithms for the

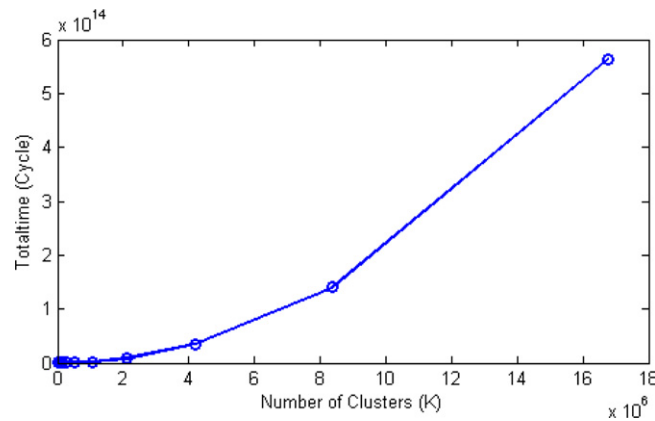


Fig. 9. Total time consumption of the algorithm vs. number of clusters K (#patterns = 32, #features = 20).

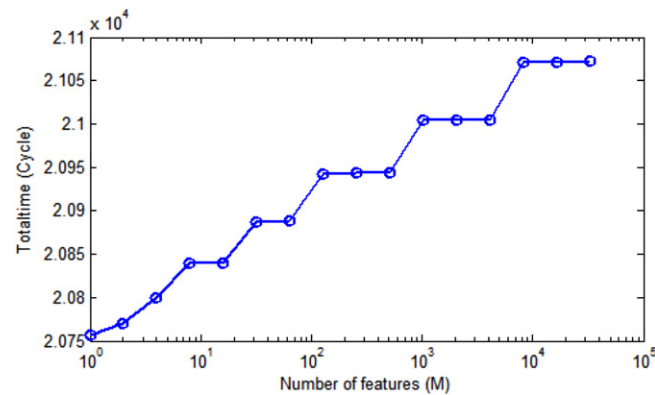


Fig. 10. Total time consumption of the algorithm vs. number of features M (#clusters=100, #patterns=200).

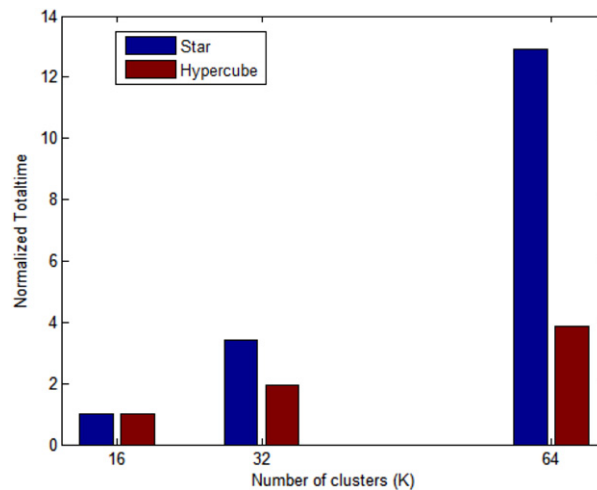


Fig. 11. The normalized total time consumption of Star-based and Hypercube-based algorithms vs. number of clusters K (#patterns = 512, #features = 20).

number of clusters being a power of 2. For each algorithm, the total execution time is normalized to the case where number of clusters is 16. In the hypercube algorithm, the number of clusters contributes linearly in the total execution time while in our algorithm it is almost exponential. It should be noted that in the hypercube algorithm, the best execution time is achieved when the number of clusters and patterns are a power of 2, while in our algorithm, the best result is given when number of clusters, patterns and features satisfy relations $N \cdot M = S!$ and $K \cdot M = T!$.

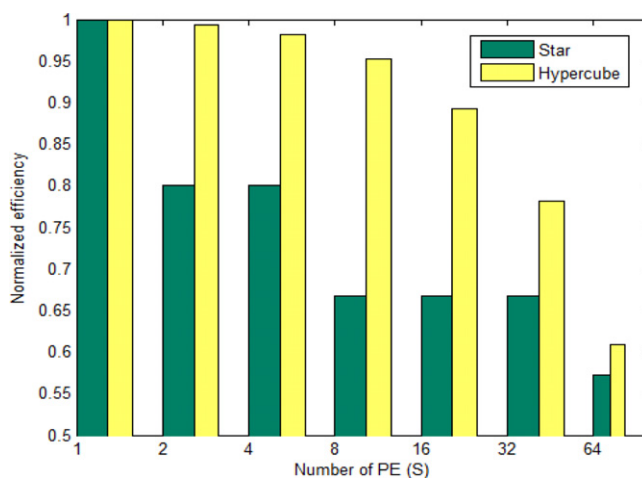


Fig. 12. The normalized efficiency of Star-based and Hypercube-based algorithms vs. number of PE (#patterns = 1024, #features = 20, #clusters = 100).

Fig. 12 shows the normalized efficiency of the proposed and hypercube algorithm. Here, efficiency is calculated based on the gained speedup divided by the number of employed processing nodes. When efficiency approaches 1, it means that the gained speedup becomes higher and when it is close to 0, it shows poor speedup gain. As can be shown in Fig. 12, when the number of processors increases, the proposed algorithm exhibits better efficiency over the hypercube algorithm. Note that for some cases in our star-based algorithm, efficiency remains unchanged as a result of the rounded inverse-factorial function.

5. Conclusions

The star graph was proposed as an attractive alternative to the hypercube topology for interconnection between processors in parallel computers. It has been extensively studied in different aspects and many algorithms have been designed for it. In this paper, a clustering algorithm for the star graph based multicomputer was presented and evaluated. This algorithm is fast and requires a little amount of memory per processing node. The algorithm completes in $O(K + S^2 - T^2)$ steps for a clustering problem of N patterns, with M features per pattern, and K clusters, where S and T are the minimum numbers such that $NM \leq S!$ and $KM \leq T!$, on an NM -node multiprocessor.

References

- [1] D.H. Ballard, C.M. Brown, *Computer Vision*, Prentice-Hall, Englewood Cliffs, NJ, 1985.
- [2] R.O. Duda, P.E. Hart, *Pattern Classification and Scene Analysis*, Wiley, New York, 1973.
- [3] K.S. Fu, *Syntactic Methods in Pattern Recognition*, Academic Press, New York, 1974.
- [4] K. Fukunaga, *Introduction to Statistical Pattern Recognition*, Academic Press, New York, 1972.
- [5] A. Rosenfeld, A.C. Kak, *Digital Picture Processing*, Academic, New York, 1982.
- [6] J.T. Tou, R.C. Gonzalez, *Pattern Recognition Principles*, Addison-Wesley, Reading MA, 1974.
- [7] S. Ranka, S. Sahni, Clustering on a hypercube multicomputer, *IEEE Transactions on Parallel and Distributed Systems* 2 (2) (1991) 71–82.
- [8] L.M. Ni, A.K. Jain, A VLSI systolic architecture for pattern clustering, *IEEE Transactions on Pattern Analysis and Machine Intelligence* 7 (1) (1985) 80–89.
- [9] K. Hwang, D. Kim, Parallel pattern clustering on a multiprocessor with orthogonally shared memory, in: *Proceedings of International Conference on Parallel Processing*, 1987, pp. 913–916.
- [10] X. Li, Z. Fang, Parallel algorithms for clustering on hypercube SIMD computers, in: *Proceedings of Conference on Computer Vision Pattern Recognition*, 1986, pp. 130–133.
- [11] M. Fazeli, H. Sarbazi-Azad, R. Farivar, Parallel clustering on the star graph, in: *Proceedings of ICA3PP*, 2005, pp. 287–292.
- [12] B. Akers, D. Harel, B. Krishnamurthy, The star graph: an attractive alternative to the n -cube, in: *Proceedings of the International Conference Parallel Processing*, 1987, pp. 393–400.
- [13] S. Akl, K. Qiu, A novel routing scheme on the star and pancake networks and its applications, *Parallel Computing* 19 (1) (1993) 95–101.
- [14] P. Fragopoulou, S. Akl, A parallel algorithm for computing Fourier transform on the star graph, *IEEE Transactions on Parallel and Distributed Systems* 5 (5) (1994) 525–531.
- [15] K. Qiu, S. Akl, Load balancing, selection and sorting on the star and pancake interconnection networks, *Parallel Algorithms & Applications* 2 (1994) 27–42.
- [16] A. Menn, A.K. Somani, An efficient sorting algorithm for the star graph interconnection network, in: *Proceedings of the International Conference Parallel Processing*, III, 1990, pp. 1–8.
- [17] H. Sarbazi-Azad, M. Ould-Khaoua, L.M. Mackenzie, S.G. Akl, A parallel algorithm for Lagrange interpolation on the star graph, *Journal of Parallel and Distributed Computing* 62 (2002) 605–621.
- [18] S. Akl, *Parallel Computation: Models and Methods*, Prentice Hall, 1997.
- [19] P. Berthone, A. Ferreira, S. Perennes, Optimal information dissemination in star and pancake networks, *IEEE Transaction on Parallel and Distributed Systems* 7 (1996) 1292–1300.