Matrix Decomposition on the Star Graph

Abdel-Elah Al-Ayyoub, Member, IEEE, and Khaled Day, Member, IEEE

Abstract—We present and evaluate, for the first time, a parallel algorithm for solving the LU decomposition problem on the star graph. The proposed parallel algorithm is of $O(N^2/n^2)$ computation complexity and uses $O(Nn)$ communication time to decompose a matrix of order $N$ on a star graph of dimension $n$, where $N \geq (n-1)!$. The incurred communication time is better than the best known results for the hypercube, $O(N \log n)$, and the mesh, $O(N \sqrt{n}l)$, each with approximately $nl$ nodes. The proposed parallel algorithm takes advantage of the attractive topological qualities of the star graph in order to reduce the communication time involved in tasks such as pivoting, row/column interchanges, and pivot row and multipliers column broadcasts.

Index Terms—Interconnection networks, matrix decomposition, parallel processing, star graphs.

1 INTRODUCTION

The star graph first appeared in [1] and has drawn a lot of attention since then. Many of its topological properties have been analyzed [13], and other related aspects have been investigated including; embedding [18], [19], [25], [29], broadcasting [17], [23], fault tolerance [6], [15], load balancing [27], routing [13], [28], star graph variants [21], and data exchange algorithms [10]. Furthermore, the solution of some problems on the star graph, such as computing Fourier transforms [14], sorting [24], [28], and ascend/descend type of divide-and-conquer algorithms [31], [32] have also been studied. The star graph has three significant advantages over the hypercube: a lower degree, a smaller diameter, and a smaller average diameter for a similar number of nodes [1], [13]. That is, the star graph offers a network with fewer links and smaller communication delays. It has a hierarchical structure and possesses many fault tolerance properties. A major practical difficulty with the star graph, however, is its poor scalability which is due to the rapid growth of its size $n!$ as a function of its dimension $n$. This issue has been recently addressed in [21], where the authors have suggested a variant of the star graph called the incomplete star, which considerably improves the scalability of the network. The incomplete star graph allows incremental scalability while preserving effective broadcasting (for some special class of incomplete star graphs).

Although there are some proposed algorithms in the literature for solving real problems such as sorting and computing Fourier transforms on the star graph, the study of algorithms for star graphs has not yet rendered any major breakthroughs. One reason could be the nonobvious mapping of data and tasks to the star graph nodes. For instance, most parallel linear algebra algorithms, as well as other types of parallel algorithms, require efficient matrix computations which, in turn, require a natural mapping of matrix elements to network nodes. Such mappings would minimize communication costs when performing matrix row or column computations. Unfortunately, natural mappings of matrix elements to star graph nodes does not seem to be simple, which has crippled the efforts of designing efficient matrix-based parallel algorithms on the star graph. Furthermore, the proposed embeddings of meshes [19], [29] and hypercubes [25] into the star graph do not allow efficient simulations of known algorithms. This paper aims at contributing toward filling this gap by presenting an efficient parallel matrix decomposition algorithm based on the star topology. We expect that future research will eventually address the design of other algorithms on this topologically attractive interconnection network.

The problem of finding solutions for systems of linear equations is often raised in many areas in engineering and science. One direct method of solution is to transform the linear system $A\vec{x} = \vec{b}$ into $LU = \vec{c}$ where $U$ is an upper triangular matrix. Then the solution vector $\vec{x}$ is obtained by back substitutions. Gaussian elimination (GE) is the standard procedure to carry out LU decomposition. Given a matrix $A$ of order $N$, the GE procedure forms a sequence of matrices $A^{(1)}$, $A^{(2)}$, ..., $A^{(N)}$ where $A^{(1)}$ is the initial matrix and $A^{(N)}$ is the desired triangular matrix. The matrix $A^{(k)}$ ($k = 2$, ..., $N$) has entries $a_{ij}^{(k)}$ defined below, and represents the equivalent linear system for which the variable $x_{k-1}$ has just been eliminated.

$$a_{ij}^{(k)} = \begin{cases} a_{ij}^{(k-1)} & \text{for } 1 \leq i < j \text{ and } 1 \leq j \leq N \\ 0 & \text{for } k \leq i \leq N \text{ and } 1 \leq j < k \\ a_{ij}^{(k-1)} - a_{ij}^{(k-1)}a_{k-1,j}^{(k-1)} & \text{for } k \leq i \leq N \text{ and } k \leq j \leq N \end{cases}$$

The above procedure will fail if any of the pivot elements $a_{1,1}^{(1)}, a_{2,2}^{(2)}, ..., a_{N,N}^{(N)}$ is zero. In practice, it is often desirable to perform partial or complete pivoting even when the pivot elements are not zero. This is necessary to ensure numerical stability on finite-digit arithmetic.

The amount of computation time required by any serial setting of GE to solve a linear system of $N$ equations is
proportional to $N^3$. Computing elements of the matrices $A^{(k)}$ requires $O(N^2)$ floating point operations. Therefore, computing $N - 1$ such matrices will be of $O(N^3)$ complexity.

Generally, there are two approaches for carrying out GE steps: These are pipelining and broadcasting. The former (well-suited for the mesh topology) allows limited pivoting and proved to be efficient when pivoting is not important or when neighboring pivoting is sufficient [9], [30]. The broadcasting approach uses broadcast primitives to exchange the pivot rows and the multipliers columns. This approach achieves better performance than pipelining when pivoting is necessary [30].

Attempting pipelining on the star graph is not appropriate because the mesh cannot be embedded with unit dilation on the star graph which reflects badly on the communication time. The broadcasting approach might appear attractive at first since the star graph has a lower all-port broadcasting cost than the hypercube. Unfortunately, there are no obvious and direct techniques for applying this approach to the star graph. In this paper, we present the first attempt towards developing efficient LU factorization algorithms for the star graph based on the broadcasting approach.

The rest of the paper is organized as follows: In Section 2, an overview of some related work is presented. In Section 3, we give an introduction about the star topology. In Section 4, we discuss cyclic matrix distribution on the star graph. Section 5 presents a parallel algorithm for LU factorization based on the star topology. Analysis of the proposed algorithm is given in Section 6. Concluding remarks are given in Section 7.

2 RELATED WORK

Gaussian elimination is the most widely used method for solving linear systems. As any other direct method of solution, Gaussian elimination is of $O(N^3)$ serial complexity. Numerous scientific computing applications require fast solution of large linear systems of equations. Therefore, the implementation of the LU decomposition on parallel computers has been extensively studied. Furthermore, LU decomposition has been used by many computer manufacturers as a benchmark [34].

Several parallel implementations of the LU decomposition have appeared in the literature. Some examples are: Block LU decomposition on the BBN TC200 [33], hypercube based parallel LU factorization [3], [8], [20], [30], matrix triangulation on mesh-connected architectures [9], parallel block LU factorization on the IBM 3090 VF/600 supercomputer [12], and pipelined ring algorithms on a ring of transputers [26]. In general, computational complexities of these parallel implementations are bounded by $O(N^3/P)$, where $N$ is the order of the linear system and $P$ is the number of processors, $N^2 \geq P$. In order to keep the Gaussian elimination time requirement minimum, it is essential to reduce the communication time. Pipelined algorithms [26], [30] alleviate the communication penalty by pipelining computations and communications at the step level and within each step. If pivoting is required, then pipelining at the step level will be very limited; hence degrading the efficiency of the algorithm. However, when pivoting is not important or when neighboring pivoting is sufficient, pipelined algorithms are more efficient than the hypercube and the star based LU factorization algorithms [9], [30].

Hypercube-based implementations of LU factorization have been investigated thoroughly [3], [4], [8], [11], [20], [30]. In these implementations, matrix elements (rows, columns, or square submatrices) are distributed over disjoint subcubes so that pivot rows and multipliers columns can be broadcasted over the disjoint subcubes. The implementation in [3] employs send-ahead optimizations in the form of broadcasts of the pivot rows and multipliers columns as soon as they are updated. These optimizations are possible only when pivoting is not required. In [30], Saad concludes that the extra connectivity of the hypercube over that of the two-dimensional mesh does not help much in improving efficiency of Gaussian elimination with pivoting. Cappello supports this claim in [8]. In [22], Lichtenstein and Johnson measured an increase of up to 20 percent in the execution time is expected to handle partial pivoting in hypercube-based implementations of LU factorization. This is due to the communication time involved in hypercube implementations of Gaussian elimination with pivoting. In this paper, we take advantage of the attractive properties of the star graph to reduce the communication time involved in pivoting, row/column interchange, and pivot and multiplier broadcasts. We start in the following section by presenting an overview of the topological properties of the star graph.

3 THE STAR TOPOLOGY

The star graph is a Cayley graph [1] with a set of vertices consisting of all permutations of $n$ symbols $(n) = \{1, 2, ... , n\}$ and with a set of $n - 1$ generators $g_2, g_3, ... , g_n$, where $g_i$ is the transposition of the symbol in the $i$th position with the symbol in the first position. For a node $u = (p_1, p_2, ... , p_n)$, $u^{(i)}$ denotes the node $(p_1, p_2, ... , p_{i-1}, p_i, p_{i+1}, ... , p_n)$, obtained by applying $g_i$ on $u$. The link connecting the two nodes $u$ and $u^{(i)}$ is labeled by the generator $g_i$ (we say a link of type $g_i$). Since there are $n!$ permutations on $n$ symbols, the $n$-star has $n!$ nodes and since there are $n - 1$ generators, the degree is $n - 1$. The $n$-star graph, $S_n$, is vertex and edge symmetric and has $\left\lceil \frac{n}{2} (n - 1) \right\rceil$ as diameter [1]. The term $d(u,v)$ denotes the minimum distance between a pair of nodes $u$ and $v$ from $S_n$.

The star graph exhibits many of the desirable properties of the hypercube while having a significantly smaller degree and diameter.

The set of nodes in an $S_n$ can be partitioned into $n$ disjoint subsets $I_1, I_2, ... , I_{n-1}$, where $I_k$ is the subset of nodes that ends with the symbol $k$. Each subgraph of $S_n$, associated with a subset $I_k$ is an $(n - 1)$-dimensional star graph denoted $S_{n-1}(k)$ [1]. Alternatively, $S_n$ can be partitioned into $n$ disjoint subsets $X_1, X_2, ... , X_n$, where $X_k$ is the subset of nodes that start with the symbol $k$.

**Proposition 1.** Any node $u$ in $X_k$ is connected to exactly one node $v$ in $I_k$ by a link of type $g_n$ in $S_n$.

Notice that there is no direct link between any pair of
nodes belonging to the same subset $X_k$. For any subset $X_k$, $k \in \langle n \rangle$, we have $|X_k| = (n-1)!$. Similarly, for any subset $I_k$, $k \in \langle n \rangle$, we have $|I_k| = (n-1)!$. A number of node ranking schemes for the star graph have appeared in the literature [5], [24], [31], [32]. Below we describe one that will be used in the subsequent sections.

**Definition 1.** Let $G_n$ be a one-to-one mapping from the set of permutations $(p_1, p_2, ..., p_n)$, $p_i \in \langle n \rangle$, $p_i \neq p_j$ for any $j \neq i$ onto the set of integers $(1, 2, ..., n!)$. For any permutation $p_1, p_2, ..., p_n$ a unique integer can be generated using the following recursive function [24].

$$G_n(p_1, p_2, ..., p_n) = \begin{cases} \lfloor \frac{n-1}{n-1} \rfloor \cdot G_{n-1}(q_1, q_2, ..., q_{n-1}) & \text{if } n = 1 \\ text{otherwise} \end{cases}$$

where $q_1, q_2, ..., q_{n-1}$ is obtained from $p_1, p_2, ..., p_n$ after dropping $p_1$ and renumbering the remaining symbols from 1 to $n-1$. For any permutation $u = (p_1, p_2, ..., p_n)$, we use $G_n(u)$ to denote $G_n(p_1, p_2, ..., p_n)$ for $1 \leq k \leq n-1$ and $q_1, q_2, ..., q_{n-k}$ is obtained after dropping the last $k$ symbols from $u$ and renumbering the remaining symbols from 1 to $n-k$.

**Proposition 2.** For any two nodes $u = (p_1, p_2, ..., p_{n-1}) \in I_k$ and $v = (q_1, q_2, ..., q_{n-1}, k+1) \in I_{k+1}$ such that $G_{n-1}(u) = G_{n-1}(v)$, we have $d(u, v) = 3$.

**Proof.** Since $p_1 = q_1$, and $G_{n-1}(u) = G_{n-1}(v)$, then $p_1 = q_1$ and $p_i = q_i + 1$ for $1 \leq i \leq n-1$, $i \neq j$. Therefore, the permutations $u$ and $v$ differ in positions $j$ and $n$. More precisely, $p_i = q_i + 1$ and $q_i = k$. Hence, if $j = 1$, then $d(u, v) = 1$; otherwise, $d(u, v) = 3$.

**Proposition 3.** For any two nodes $u = (k, p_2, p_3, ..., p_n) \in X_k$ and $v = (k+1, q_2, q_3, ..., q_n) \in X_{k+1}$ such that $G_{n-1}(u^{(n)}) = G_{n-1}(v^{(n)})$, we have $d(u, v) = 1$.

**Proof.** Since $p_1 = q_1 - 1$ and $G_{n-1}(u^{(n)}) = G_{n-1}(v^{(n)})$, then $p_1 = q_1$ and $p_i = q_i + 1$ for $2 \leq i \leq n$, $i \neq j$. More precisely, $p_1 = q_1 + 1$ and $q_1 = k$. Therefore, $u^{(n)} = v^{(n)}$, and the result follows.

**Proposition 4.** For any two nodes $u = (k, p_2, p_3, ..., p_n) \in X_k$ and $v = (k, q_2, q_3, ..., q_n) \in X_n$ we have $d(u, v) \geq 3$.

**Proof.** Since $u, v \neq q_i$ and $p_1 = q_1$, then there should be at least two positions $i$ and $j$ (other than the first) such that $p_j \neq q_i$ and $p_j \neq q_j$. Hence, $d(u, v) \geq 3$.

### 4 Cyclic Matrix Distribution on the Star Graph

The distribution of the matrix elements onto a set of processors is a key factor for effective parallel matrix computation. In particular, parallel broadcasting across rows and columns should be effectively supported by the matrix distribution techniques. Such techniques are easily achievable on meshes and hypercubes [3], [4], [8], [9]. A simple matrix distribution on the hypercube can be done by partitioning the $n$-bit binary addresses of the $n$-cube into two equal parts. This partitions the $n$-cube into $2^{n/2}$ disjoint subcubes of dimension $n/2$. Thus, matrix elements can be distributed over these disjoint subcubes such that elements of a same row (resp. column) reside in the same subcube.

Adopting this technique for matrix distribution in the star graph is not obvious. It is apparently unachievable to define a distribution that allows us to use star-based broadcasting in both directions (row and column). In this section, we discuss cyclic matrix distribution techniques that get around this problem. Similar techniques were used in [2], [24] to perform sorting and various computational geometric problems on the star graph. Let $\hat{A} = [a_{ij}]$, $i, j \leq n$, be the set of elements in an $n \times n$ matrix and let $\hat{V}$ be the set of nodes in $S_n$, where $n \geq (n-1)!$.

**Definition 2.** The star cyclic matrix distribution is a function 

$$SCMD: \hat{A} \rightarrow \hat{V} \text{ given by } SCMD(a_{ij}) = v, \text{ such that } v \in I_R$$

and $G_{n-1}(v) = C$, where $R = [(i-1) \mod n] + 1$ and $C = [(j-1) \mod (n-1)!] + 1$. Such a node is denoted by $P_{RC}$.

The function $SCMD$ distributes the matrix rows over the set of $n$ substars cyclically. Within each substar, the row elements are also distributed cyclically over its nodes. Formally speaking, for each $R$ and $C$, $1 \leq R \leq n$ and $1 \leq C \leq (n-1)!$, let $\lambda_R$ be the largest integer such that $R + \lambda_R n \leq N$ and let $\mu_C$ be the largest integer such that $C + \mu_C(n-1)! \leq N$. A node $P_{RC}$ will be assigned the submatrix $A_{Nn,Rn/(n-1)!} = [a_{ij}]$, $i = R$, $R + n, R + 2n, ..., R + \lambda_R n$, and $j = C, C + (n-1)!$, $C + 2(n-1)!$, ..., $C + \mu_C(n-1)!$. This type of matrix distribution is called cyclic data distribution and is known to achieve good load balancing and processor utilization [22]. The reader is referred to [4], [22] for further discussion on different matrix distribution techniques. Fig. 1 shows an example of distributing a $6 \times 6$ matrix onto an $S_4$ graph using the $SCMD$ function. Notice that using $SCMD$, each matrix row resides in an $S_{n-1}$ substar.

![Fig. 1. Matrix distribution using SCMD on $S_4$.](image-url)

**Definition 3.** The linear array cyclic matrix distribution is a function $LCMD: \hat{A} \rightarrow \hat{V}$ given by $LCMD(a_{ij}) = v$ such that $v \in X_R$ and $G_{n-1}(v^{(n)}) = C$, where $R = [(i-1) \mod n] + 1$ and $C = [(j-1) \mod (n-1)!] + 1$. Such a node is denoted by $P_{LC}$.
The function $\text{LCMD}$ distributes the $N$ matrix rows over the sets $X_1, X_2, \ldots, X_n$ cyclically. Within each set, the row elements are also distributed cyclically over the nodes of this set. Formally speaking, for each $R$ and $X_i$, $1 \leq R \leq n$ and $1 \leq i \leq (n - 1)!$, let $\lambda_R$ be the largest integer such that $R + \lambda_R n \leq N$ and let $\mu_R$ be the largest integer such that $\lambda_R + \mu_R (n - 1)! \leq N$ (notice that $R$ and $C$ defined for the function $\text{SCMD}$ differ from $R$ and $C$ defined for $\text{LCMD}$). A node $P_{RC}$ will be assigned the submatrix $A_{\lambda_i \mu_i N / (n - 1)!} = [a_{ij}]$, $i = R, R + n, R + 2n, \ldots, R + \lambda_R n$ and $j = C, C + (n - 1)!$, $C + 2(n - 1)!$, $\ldots$, $C + \mu_R (n - 1)!$. Fig. 2 shows an example of distributing a $6 \times 6$ matrix onto an $S_4$ using the $\text{LCMD}$ function. Notice that using $\text{LCMD}$, each matrix column resides in a set of nodes connected in a linear array of $n$ nodes. The following observations about the functions $\text{SCMD}$ and $\text{LCMD}$ can be easily verified:

1) Using $\text{SCMD}$, elements of the same row are stored in the same $S_{n-1}$. Therefore, the farthest distance between any pair of nodes holding elements of the same row is $\left\lceil \frac{2}{3}(n - 2) \right\rceil$.

2) Using $\text{LCMD}$, consecutive column elements are stored in nodes at a distance of one (the column elements are connected in a linear array). The farthest distance between any pair of nodes holding elements of the same column is $n - 1$ (Proposition 3).

3) Using $\text{SCMD}$, consecutive column elements are stored in nodes at a farthest distance of three (Proposition 2).

4) Using $\text{LCMD}$, elements of the same row are stored in nodes at a closest distance of three (Proposition 4).

5) Any node $u = (p_u, p_2, \ldots, p_u)$ denoted by $P_{RC}$ in $\text{SCMD}$, is connected to the node $u''$ denoted by $P_{RC}$ in $\text{LCMD}$, via a link of type $g_{uu'}$, where $R = p_u$, $C = G_{n-1}(u)$, $R = p_u$, and $C = G_{n-1}(u'')$ (Proposition 1).

6) A node $u$ denoted by $P_{RC}$ in $\text{LCMD}$, is directly connected to a node $v$ denoted by $P_{RC}$ in $\text{LCMD}$ with a link of type $g_{uu''}$ in $S_n$ where $\alpha(i)$ is the position occupied by the symbol $i$ in $u$. Similarly, a node $P_{RC}$ in $\text{LCMD}$ is directly connected to a node $P_{RC}$ in $\text{LCMD}$ with a link of type $g_{uu''}$ in $S_n$ (Proposition 3).

The function $\text{SCMD}$ distributes the matrix rows over $n$ disjoint $S_{n-1}$ substars. Therefore, subcolumn broadcasts can be performed simultaneously in these subgraphs. The function $\text{LCMD}$ distributes the matrix columns on the $(n - 1)!$ disjoint sets of nodes where each set of nodes forms a linear array. Therefore, simultaneous subrow broadcasts are possible. Broadcasting a matrix row (resp. column) in $\text{LCMD}$ (resp. $\text{SCMD}$) requires $O(n)$ broadcasting steps. Such an optimal broadcasting algorithm on the star graph can be obtained by using all-port broadcasting on a minimum height spanning tree rooted at the source node in a maximum number of steps equal to the diameter of the graph [13].

In summary, $\text{LCMD}$ allows efficient row broadcasts and $\text{SCMD}$ allows efficient column broadcasts. Switching between $\text{SCMD}$ and $\text{LCMD}$ requires a single submatrix-exchange step (see observation 5 above). However, with $\text{SCMD}$, both row and column broadcasts are possible and there will be no need for the submatrix-exchange step. Row broadcast is more expensive in $\text{SCMD}$ because rows will travel across a dilation three “simulated” linear array (see Proposition 2). Furthermore, the function $\text{SCMD}$ can be used to distribute either rows or columns to the disjoint $S_{n-1}$ subgraphs (simply switch the expressions for $R$ and $C$ in Definition 2). In the subsequent sections we present parallel LU decomposition algorithms based on $\text{SCMD}$ and $\text{LCMD}$.

5 LU Decomposition on the Star Topology

Given a dense matrix $A$ of order $N$, Gaussian elimination can be used to decompose this matrix by generating $N - 1$ matrices $A^{(0)}, \ldots, A^{(n)}$ as defined in the introduction section. The steps involved in computing these matrices are given below:

for $k = 1$ to $N - 1$ do

\hspace{1cm} Partial pivoting: interchange the rows of $A^{(k)}$ such that $\left| a_{ik}^{(k)} \right| = \max_{k \leq l \leq n} \left| a_{lk}^{(k)} \right|

\hspace{1cm} Task \{ T_{ik,k} \} \equiv \{ \text{Partial pivoting} \} \quad | k \leq l \leq k \leq n \}

\hspace{1cm} Complete pivoting: interchange rows and columns of $A^{(k)}$ such that $\left| d_{ik}^{(k)} \right| = \max_{k \leq l \leq k \leq n} \left| d_{lj}^{(k)} \right|

\hspace{1cm} endfor

for $i = k + 1$ to $N$ do

\hspace{1cm} Task $\{ T_{ik,k} \} \equiv \{ \text{Compute multipliers} \} d_{ik}^{(k)} = a_{ik}^{(k)}/d_{kk}^{(k)}$

endfor

for $j = k + 1$ to $N$ do

\hspace{1cm} for $j = k + 1$ to $N$ do

\hspace{2cm} Task $\{ T_{ij,j} \} \equiv \{ \text{Eliminate} \} a_{ij}^{(k)} = d_{ij}^{(k)} \cdot a_{ij}^{(k)}$

endfor

endfor

In our approach of parallelizing LU decomposition, elements of the matrix $A$ are distributed in a cyclic manner over the $n!$ processors of $S_n$. Each processor will hold one or more matrix elements. Without pivoting, processors can compute multipliers $\{ T_{ik,j} \}$ and perform eliminations $\{ T_{ij,j} \}$.
on their domain whenever the proper multipliers and pivot elements are available.

Partial pivoting requires the following steps to be performed: The set of processors holding elements of the kth column should find \( a_{r,k}^{(k)} \) such that \( |a_{r,k}^{(k)}| = \max_{r \leq i,j \leq N} |a_{i,j}^{(k)}| \), and then set the processors holding elements of the kth and the rth rows should interchange the relevant subrows.

With complete pivoting, the set of processors holding elements of the submatrix \([ a_{ij}^{(k)} ]\), \( k \leq i,j \leq N \), should find \( a_{r,s}^{(k)} \) such that \( |a_{r,s}^{(k)}| = \max_{r \leq i,j \leq N} |a_{i,j}^{(k)}| \), and then set the processors holding elements of the kth and the rth rows should interchange the relevant subrows.

The set of processors holding elements of the submatrix \([ a_{ij}^{(k)} ]\), \( k \leq i,j \leq N \), should find \( a_{r,s}^{(k)} \) such that \( |a_{r,s}^{(k)}| = \max_{r \leq i,j \leq N} |a_{i,j}^{(k)}| \), and then set the processors holding elements of the kth and the rth rows should interchange the relevant subrows.

With complete pivoting, the set of processors holding elements of the kth and the rth rows should interchange the relevant subrows and the set of processors holding elements of the kth and the sth columns should interchange the relevant subcolumns.

At any point of time, a processor can be in any of the following states:
1. broadcasting pivot subrow or multipliers subcolumn,
2. eliminating a submatrix,
3. involved in determining the new pivot row,
4. exchanging submatrices,
5. waiting for a pivot subrow and/or a multipliers subcolumn,
or
6. idle holding final matrix elements.

The algorithm requires \( N - 1 \) steps to decompose a matrix of order \( N \). At the kth step, processors tasks can be described as follows:

- Processor \( P_{R,C} \), where \( R_k = [(k-1) \mod n] + 1 \) and \( 1 \leq C \leq (n-1)! \), should broadcast the pivot subrow \([ a_{ik}^{(k)} ]\) to all processors in the set \( \{ P_{X,i} \mid 1 \leq X \leq n \} \).
- Each processor \( P_{R,C} \), where \( 1 \leq R \leq n \) and \( 1 \leq C \leq (n-1)! \), excluding those holding the pivot subrows should wait until the pivot subrow is received and then exchange its submatrix with that of \( P_{R,C} \) using the link \( g_r \).
- Processor \( P_{R,C} \), where \( 1 \leq R \leq n \) and \( C_k = [(k-1) \mod (n-1)!] + 1 \), should compute the multipliers subcolumn \( (T_{1,j}) \), where \( i = R, R + n, R + 2n, ..., R + \lambda_R n \), and broadcast it in \( S_{n-1}(R) \).
- Processor \( P_{R,C} \), where \( 1 \leq R \leq n \) and \( C_k \), should broadcast the pivot subrow \([ a_{ik}^{(k)} ]\) to all processors in the set \( \{ P_{X,i} \mid 1 \leq X \leq n \} \).

When partial pivoting is required, the kth pivot row is determined by the set of processors

\[
\{ P_{L,C} \mid 1 \leq L \leq n \} \text{ and } C_k = [(k-1) \mod (n-1)!] + 1
\]

These processors perform an “exchange-max” procedure. At the end of this procedure, each processor will have a copy of the index of the desired pivot row, say \( \text{imax} \). If the rows \( k \) and \( \text{imax} \) are in the same set \( X_{R_k} \), where \( R_k = [(k-1) \mod n] + 1 \), then each processor in \( X_{R_k} \) swaps subrows \( a_{ij} \) and \( a_{\text{imax,j}} \), \( j = 1, C + (n-1)! \), \( C + 2(n-1)! \), ..., \( C + \mu_c(n-1)! \), and \( k \leq i,j \leq N \). Processors holding the multipliers subcolumns need not to wait and may proceed with elimination.

The algorithm executed by each node \( u = (p_1, p_2, ..., p_n) \) in the \( S_n \) graph is outlined in Fig. 3. The matrix is initially distributed using the function \( \text{LCMD} \). The algorithm can be easily adapted if only \( \text{SCMD} \) is to be used for matrix distribution, or if the function \( \text{SCMD} \) is to be used to distribute columns (instead of rows) to the disjoint \( S_{n-1} \) substars. The procedures \( \text{broadcast_linear_array} \) and \( \text{broadcast_star} \) perform broadcasting in the linear array and the star graph, respectively. Optimal all-port broadcasting on both topologies would be of \( O(n) \) time complexity, where \( n \) is the dimension of the graph. The procedure \( \text{exchange_submatrices} \) is used to exchange submatrices between \( P_{R,C} \) and \( P_{L,C} \). This can be achieved by issuing an asynchronous receive from \( P_{R,C} \) (resp. \( P_{L,C} \)) followed by a synchronous send to \( P_{L,C} \) (resp. \( P_{R,C} \)). When a processor executes the procedure \( \text{wait} \), it keeps checking all ports until the specified value is received.

---

**Algorithm LFI Decomposition**

The following code is executed by the node \( u = (p_1, p_2, ..., p_n) \): 1. Compute \( R = p_{u}, C = G_{n-1}(u), R = p_{1}, A = G_{n-1}(u^{(0)}) \)
2. For \( k = 1 \) to \( N - 1 \) do
   a. If \( R = (k-1) \mod n + 1 \) then
      b. \( C = [(k-1) \mod (n-1)!] + 1 \)
      c. Execute \( (T_{1,k}) \)
      d. If \( R = R_k \) then broadcast_linear_array \( ([a_{ik}^{(k)}] j = C, C + (n-1)! \), \( C + 2(n-1)! \), ..., \( C + \mu_c(n-1)! \), and \( k \leq j \leq N \), \( \{ P_{X} \} \)
      e. If \( R = R_k \) then wait \( ([a_{ik}^{(k)}] j = C, C + (n-1)! \), \( C + 2(n-1)! \), ..., \( C + \mu_c(n-1)! \), and \( k \leq j \leq N \), \( \{ P_{X} \} \)
      f. Execute \( (T_{1,k}) \)
      g. If \( C = C_k \) then
         i. Execute \( (T_{1,k}) \), \( i = R, R + n, R + 2n, ..., R + \lambda_R n \) and \( k < i \leq N \), \( \text{broadcast_star} ([a_{ik}^{(k)}] j = R, R + n, R + 2n, ..., R + \lambda_R n \) and \( k < i \leq N \), \( S_{n-1}(R) \)
      h. Execute \( (T_{1,k}) \), \( i = R, R + n, R + 2n, ..., R + \lambda_R n \) and \( k < i \leq N \) and \( S_{n-1}(R) \)
   f) end

---

Fig. 3. Star-based matrix decomposition algorithm.
Task (T_{k,3}): Partial pivoting

if C = C_k then
    let M_1 = -1 and imax_1 = 1
    find imax_1 such that
    \[ |a_{\text{imax}_1,1}| = \max \left\{ |a_{i,j}| \mid i = R, R + n, R + 2n, \ldots, R + \lambda_2 n \right\} \]
    if (k ≤ imax_1 ≤ N) then M_1 = |a_{\text{imax}_1,1}|
    for j = 1 to n - 1 do
        if R = 1 then M_2 = -1 and imax_2 = 1
        else asynchronously receive (M_2, imax_2) from P_{E-1,E} using
        the link \( s_{\text{a(E-1)}} \)
        if R = n then M_3 = -1 and imax_3 = 1
        else asynchronously receive (M_3, imax_3) from P_{E-1,E} using
        the link \( s_{\text{a(E-1)}} \)
        if R ≠ 1 then synchronously send (M_1, imax_1) to P_{E-1,E} using
        the link \( s_{\text{a(E-1)}} \)
        if R ≠ n then synchronously send (M_2, imax_2) to P_{E-1,E} using
        the link \( s_{\text{a(E-1)}} \)
        find h such that \( M_h = \max |M_i| \mid i = 1, 2, 3 \]
        let M_1 = M_h and imax_1 = imax_h
    endfor
    broadcast_star (imax_1, S_{n-1}(R))
else wait (imax_1)
    V_k = [(imax_1 - 1) mod n] + 1
    if R = V_k = R_k then Swap_subrows (k, imax_1)
    elseif R = V_k ≠ R_k then Interchange_subrows (k, imax_1, R_k)
    elseif R ≠ V_k = R_k then Interchange_subrows (imax_1, k, V_k)
end Task (T_{k,3})

Task (T_{k,4}): Complete pivoting

let M_1 = -1, imax_1 = 1, and jmax_1 = 1
find imax and jmax such that
\[ |a_{\text{imax, jmax}}| = \max \left\{ |a_{i,j}| \mid i = R, R + n, R + 2n, \ldots, R + \lambda_2 n \right\} \]
\[ j = C, \widetilde{C} + (n - 1), \widetilde{C} + 2(n - 1), \ldots, \widetilde{C} + \mu_{\widetilde{C}}(n - 1) \]
if (k ≤ imax_1, jmax_1 ≤ N) then let M_1 = |a_{\text{imax}_1, jmax_1}|
for j = 1 to \( \frac{n}{2} (n - 1) \) do
    asynchronously receive (M_j, imax_j, jmax_j), i = 2, ..., n, from all
    neighbors in \( S_n \)
    synchronously send (M_j, imax_j, jmax_j) to all neighbors in \( S_n \)
    wait for asynchronous receive to complete
    find h such that \( M_h = \max |M_i| \mid i = 1, 2, 3 \]
    let M_1 = M_h, imax_1 = imax_h, and jmax_1 = jmax_h
endfor
R_{V_j} = [(imax_1 - 1) mod n] + 1
C_{V_j} = [(jmax_1 - 1) mod (n - 1)] + 1
if (R_{V_j} = R_k) and (C_{V_j} = C_k) then
    if R = R_k then Swap_subrows (k, imax_1)
    elseif R = C_k then Swap_subcolumns (k, jmax_1)
    elseif (R_{V_j} = R_k) and (C_{V_j} ≠ C_k) then
        exchange_submatrices (R, C, R_k, C_{V_j})
    elseif C = C_{V_j} then Interchange_subcolumns (k, jmax_1, C_k)
    elseif C = C_k then Interchange_subcolumns (jmax_1, k, C_{V_j})
        exchange_submatrices (R, C, R_k, C_{V_j})
    elseif (R_{V_j} ≠ R_k) and (C_{V_j} = C_k) then
        if C = C_k then Swap_subcolumns (k, jmax_1)
        elseif R = R_{V_j} then Interchange_subrows (k, imax_1, R_k)
        elseif R = R_k then Interchange_subrows (imax_1, k, R_{V_j})
            exchange_submatrices (R, C, R_k, C_{V_j})
    elseif R = R_{V_j} then Interchange_subrows (k, imax_1, R_k)
        exchange_submatrices (R, C, R_k, C_{V_j})
    elseif C = C_{V_j} then Interchange_subcolumns (jmax_1, k, C_{V_j})
        exchange_submatrices (R, C, R_k, C_{V_j})
    elseif R = R_{V_j} then Interchange_subrows (k, imax_1, R_k)
        exchange_submatrices (R, C, R_k, C_{V_j})
end Task (T_{k,4})

Fig. 4. Star-based partial pivoting algorithm.

The procedures Swap_subrows and Interchange_subrows are used to swap rows within the same processor and between two different processors in a linear array, respectively. One possible coding for the procedure Interchange_subrows is as follows: The procedure receives two row indices i and j and a destination processor “number” in the associated linear array. It then issues an asynchronous receive to get the subrow i from the destination processor, followed by a synchronous send to pass the subrow j to the same destination processor, and then blocks until the asynchronous receive is complete.

In the case of complete pivoting, all processors in the \( S_n \) graph will perform “exchange-max” to find imax and jmax such that \( |a_{\text{imax, jmax}}| = \max \left\{ |a_{i,j}| \mid k ≤ i, j ≤ N \right\} \). Then, the kth pivot row is determined by swapping the rows k and imax and swapping the columns k and jmax. This is achieved as follows: If the rows k and imax are in the same set \( X_{R_k} \), should swap subrows \( a_{k,j} \) and \( a_{ \text{imax}_1, j} = \widetilde{C} + (n - 1)!, \widetilde{C} + 2(n - 1), \ldots, \widetilde{C} + \mu_{\widetilde{C}}(n - 1)! \). If the rows k and imax are in two different sets \( X_{R_k} \) and \( X_{R'} \) where \( V_k = [(imax - 1) mod n] + 1 \), then each processor pair \( P_{R_k,E} \) and \( P_{R',E} \) where \( 1 ≤ \widetilde{C} ≤ (n - 1)! \)

Fig. 5. Star-based complete pivoting algorithm.
should interchange subrows $a_{ij}$ and $a_{i\max,j}$. Interchanging the columns $k$ and $j\max$ requires that the matrix be redistributed using SCMD. A star-based complete pivoting algorithm is given in Fig. 5.

Finding $i\max$ and $j\max$ such that $|a_{i\max,j\max}| = \text{Max}(|a_{ij}|)$ for all processors in $S_n$ execute the exchange-max procedure simultaneously. This procedure repeatedly alternates between two phases. In the first phase, processors initiate asynchronous receive from all neighbors and block on synchronous send to all neighbors in order to exchange local extremes. In the second phase, processors update their local extremes that will be exchanged in the next iteration. It can be simply proved that, after a number of iterations equal to the diameter of the graph, the exchange-max procedure will terminate with each processor having a copy of the maximum. An $n$-cube based exchange-max procedure requires $n$ steps to terminate successfully. In each iteration, a processor updates the local extremes that will be exchanged in the next iteration. It can be simply proved that, after a number of iterations equal to the diameter of the graph, the exchange-max procedure will terminate with each processor having a copy of the maximum. An $n$-cube based exchange-max procedure requires $n$ (the graph diameter) iterations to terminate successfully. In each iteration, a processor receives a local extreme from one neighbor, updates its local extreme, and resends the updated local extreme to the same neighbor.

Allowing more than one simultaneous send/receive will not improve the performance of the $n$-cube based exchange-max procedure. This is due to the fact that each processor does not have to wait for a neighbor to complete its send/receive operation before it can initiate its own send/receive operation.

In 2D matrix distribution, $P$ processors are arranged as an $R \times C$ grid [4], [30]. For the $h$-cube, this can be done by partitioning the $h$-bit binary addresses into two parts ($R = 2^m$ and $C = 2^{c-m}$, $1 \leq m \leq h$). For an $n$-star, the decomposition is done using SCMD and LCMD with $R = n$ and $C = (n-1)!$.

Let $\rho_r(1 \leq r \leq R)$ denote the set of processors in the $R \times C$ grid along the $r$th row, and let $\chi_c(1 \leq c \leq C)$ denote the set of processors along the $c$th column. Given a matrix $A$ of order $N$ distributed over the $P$ processors using a given matrix distribution function, a broadcasting-based LU decomposition algorithm requires $N-1$ factorization steps to decompose the matrix $A$. In the $k$th step, the following tasks are performed:

- $(T_{ik})$
- $C$ simultaneous broadcasts in the $\chi_c$ sets ($1 \leq c \leq C$) with message length equal to $N/C$.
- $N/R$ sequential $(T_{ij})$.
- $R$ simultaneous broadcasts in the $\rho_r$ sets ($1 \leq r \leq R$) with message length equal to $N/R$.
- $N^2/P$ sequential $(T_{ij})$.

Thus, the maximum execution time of a single factorization step, $\tau_s$, is given by:

$$\tau_s = \varphi(T_{ik}) + \varphi(\chi_c) + \frac{N}{P} \varphi(T_{ij}) + \varphi(\beta_r) + \frac{N^2}{P} \varphi(T_{ij})$$

where

- $\varphi(T_{ik})$ is the total time required to perform $(T_{ik})$ which is equal to the time needed to locate the pivot row plus the time needed to route subrows/subcolumns.
- $\varphi(\chi_c)$ is the time required to broadcast a pivot subrow of $N/C$ elements in $\chi_c$.
- $\varphi(\rho_r)$ is the time required to broadcast a multipliers subcolumn of $N/R$ elements in $\rho_r$.
- $\varphi(T_{ij})$ is the time required to perform $(T_{ij})$.
- $\varphi(T_{ij})$ is the time required to perform $(T_{ij})$.

The broadcasting-based LU factorization algorithms overlap communication and computation in two levels: intrastep and interstep. Overlapping within each step does not reduce the overall execution time because the time needed to perform a single step depends primarily on the last set $\rho_r$ that receives the pivot row (no matter how soon the other sets start computing and broadcasting the multipliers column). However, interstep overlapping affects the overall execution time as described below.

Interset overlapping can be achieved based on the fact that locating the $k$th pivot row and computing the $k$th multipliers column can start as soon as the $(k-1)$st multipliers column is received. Formally speaking, a new step can start every

$$\tau_B = \varphi(T_{ik}) + \varphi(\chi_c) + \frac{N}{P} \varphi(T_{ij}) + \varphi(\beta_r) + \frac{N^2}{P} \varphi(T_{ij})$$

The structure of the underlying network. In fact, the different methods of static matrix distribution (row, column, block, cyclic, reflection, etc.) will not reduce the overall processor idle time; rather, the sum of the processors idle time is redistributed [8].
time units, where \( \Phi(\beta) \) is the time passed until the set of processors holding the next multipliers column receive the previous multipliers subcolumn, \( 0 \leq \Phi(\beta) \leq \Phi(\rho) \). Notice that in cyclic and reflection distribution methods, \( \Phi(\beta) > 0 \) because elements of two consecutive columns cannot be held by the same processor (trading communication cost with load balancing). Taking this overlapping into consideration and given that each step needs \( t_s \) time units, the estimated execution time \( t \) for a broadcasting-based parallel LU decomposition algorithm is given by

\[
t = (N - 2)t_s + t_s \tag{2}
\]

From the above analysis, we conclude that the computation time of a broadcasting-based LU decomposition algorithm is proportional to \( O(N^3 / \rho) \) where \( N^2 \geq \rho \). Therefore, the computation time needed for each of the hypercube, the mesh, and the star based algorithms is the same. However, the communication time requirement of a star-based implementation is better than both the hypercube and the mesh as justified below.

The optimal number of broadcasting steps needed to route the pivot rows and multipliers columns on a wraparound mesh of \( n! \) nodes is achieved using a “square” wraparound mesh \( M(\sqrt{n!}, \sqrt{n!}) \). This will optimize the time measures \( \Phi(\chi) \) and \( \Phi(\rho) \) as well as the message length. In this setting, a mesh-based LU decomposition algorithm using the broadcasting approach requires \( O(\sqrt{n!}) \) communication steps. The message length communicated in each step is proportional to \( O(N / \sqrt{n!}) \). The smallest hypercube of size \( n! \) is \( Q_{\log n!} \). An obvious subcube partitioning that achieves the best broadcasting results is to divide the \( \log n! \) address bits of the \( Q_{\log n!} \) into two equal parts. The required pivot row and multipliers column broadcasting will then be on \( Q_{\log n!/2} \) subcubes. Therefore, the required number of communication steps is \( O(\log n!) \) and the message length is \( O(N / \sqrt{n!}) \).

The proposed algorithm for the star-based LU decomposition achieves better communication performance than both the hypercube and the mesh. The number of communication steps required to broadcast a multipliers subcolumn in \( S_{n-1} \) and a pivot subrow in a linear array of \( n \) nodes are both \( O(n) \). One should notice that two submatrix interchanges are needed to perform matrix redistribution in the case of partial pivoting, and four submatrix interchanges are needed in the case of complete pivoting (see Figs. 4 and 5). Each submatrix interchange requires one parallel communication step and the message length is \( O(N^2 / n!) \). Submatrix-interchange is necessary to switch from SCMD to LCMD and vice versa. If this step is ignored, then pivot row broadcasts will be more expensive, since they will travel across a dilation three “simulated” linear array (see Proposition 2). However, broadcasting pivot subrows will still require no more than \( O(n) \) communication steps. The message length communicated in broadcasting multipliers subcolumns is \( O(N/n) \), which is larger than that of the hypercube and the mesh. However, the message length communicated in broadcasting pivot subrows is \( O(N(n - 1)/n!) \), which is smaller than that of the hypercube and the mesh. The question now is which topology offers the best overall performance considering both the number of communication steps and the message length.

A model commonly used to describe the communication time required for broadcasting a message of length \( M \) in a graph of diameter \( \delta \) is \( \delta(b + aM) \), where \( b \) is the message latency and \( a \) is the unit transmission cost [3], [8], [11]. Another tighter bound on the cost of multiple-port one-to-all broadcasting in a vertex-transitive graph of degree \( \Delta \) and diameter \( \delta \) is given by \( (\sqrt{Ma / [b\Delta + \sqrt{\delta - 1}^2]} b \) [16]. Using the latter model, we plot in Fig. 6 the estimated execution time given by (2) for each of the hypercube, the mesh, and the star broadcasting-based LU decomposition algorithms with partial pivoting. The parameters \( a \) and \( b \) are set to \( 1 \) \( \mu \)s and \( 1,000 \) \( \mu \)s, respectively. Existing parallel architectures offer such speeds [16]. The figures suggest that the star-based algorithm incurs a lower cost than the hypercube and the mesh topologies when pivoting is necessary. Notice that the cost for the mesh increases for larger network sizes, which is due to the dominance of the increase in the communication cost over the decrease in the computation cost.

Although the number of communication steps required by the star-based LU factorization algorithm, \( O(n) \), is much less than the hypercube-based algorithm, \( O(\log n!) \sim O(n \log n) \),
Fig. 6 suggests a little difference in the execution time. This is mainly attributed to the high message lengths involved in the star-based algorithm. Reducing the message length by relaxing the restriction $N ≥ (n − 1)!$ will leave a large number of the processors idle; hence, trading load balancing with communication cost. Another shortcoming of the proposed algorithm is that it needs a submatrix-interchange step in order to switch between $SCMD$ and $LCMD$. This step limits interstep overlapping and ignoring it will add a constant factor to the row communication cost.

7 Conclusion
The major contribution of this paper is the design and evaluation, for the first time, of a parallel LU decomposition algorithm based on the star topology. Parallel algorithms for carrying out partial and complete pivoting are also discussed. Compared to the hypercube and the mesh-based LU decomposition with pivoting [3], [4], [8], [9], [20], [30], the star-based parallel LU decomposition presented in this paper is more efficient for at least the following two reasons: First, broadcasting pivot rows and multipliers columns is faster on the star graph than the hypercube. When communication along all channels can take place simultaneously, an optimal broadcasting in the $S_n$ graph requires $O(n)$ steps which is much lower than $O(\log n!)$ steps that are required by an optimal broadcasting in a hypercube or $O(\sqrt{n!})$ steps that are required by an optimal broadcasting in a mesh. Second, the exchange of rows/columns can be done more efficiently between star interconnected processors since the average distance for the star graph is less than the hypercube and the mesh. Finally, we have presented timing models for estimating and comparing computation and communication time of broadcasting-based LU decomposition on each of the star graph, the hypercube, and the mesh topologies.

References

Abdel-Elah Al-Ayyoub received his BSc degree in computer science in 1986 from Yarmouk University, Jordan. He then joined the Middle East Technical University, Turkey, where he obtained his MS and PhD degrees in computer engineering in 1987 and 1992, respectively. His areas of interest include interconnection networks, parallelizing compilers, and the design of parallel algorithms. Dr. Al-Ayyoub is currently an assistant professor in the Computer Science Department at Sultan Qaboos University, Oman. He is a member of the ACM, IEEE, IASTED, and the IAMCM societies.

Khaled Day received an engineering degree in computer science from the University of Tunis in 1986 and the MSc and PhD degrees from the University of Minnesota in 1989 and 1992, respectively. He joined the faculty of the Computer Science Department of the University of Bahrain in 1992. In September 1996, he moved to the Computer Science Department of Sultan Qaboos University in Oman. His areas of interest include interconnection networks, parallel computing, distributed systems, and computer networks. He is a member of the IEEE and the ACM.