International Journal of High Speed Computing, Vol. 9, No. 2 (1997) 85–99

PARALLEL SOLUTION OF DENSE LINEAR SYSTEMS ON THE k-ARY n-CUBE NETWORKS

ABDEL-ELAH AL-AYYOUB and KHALED DAY
Department of Computer Science, Sultan Qaboos University
P. O. Box 36, Al-Khod, Postal Code 123
Sultanate of Oman
e-mail: *ayyoub@acm.org*
e-mail: *kday@squ.edu.om*

Received March 1996

ABSTRACT
In this paper a parallel algorithm for solving systems of linear equation on the k-ary n-cube is presented and evaluated for the first time. The proposed algorithm is of $O(N^3/k^n)$ computation complexity and uses $O(Nn)$ communication time to factorize a matrix of order N on the k-ary n-cube. This is better than the best known results for the hypercube, $O(N \log k^n)$, and the mesh, $O(N\sqrt{k^n})$, each with approximately k^n nodes. The proposed parallel algorithm takes advantage of the extra connectivity in the k-ary n-cube in order to reduce the communication time involved in tasks such as pivoting, row/column interchanges, and pivot row and multipliers column broadcasts.

Keywords: Interconnection topologies, k-ary n-cube, linear systems, parallel computing.

1. Introduction. Numerous scientific computing applications require fast solution of large linear systems of equations. One direct method of solution is to transform the linear system to a triangular system where the solution are obtained by backward/forward substitutions. Gaussian elimination (GE) is the standard procedure to carry out matrix triangulization to solve the system $A\mathbf{x} = \mathbf{b}$. This procedure forms a sequence of matrices $A^{(1)}, A^{(2)}, \ldots, A^{(N)}$ where $A^{(1)}$ is the initial matrix and $A^{(N)}$ is the desired triangular matrix. The matrix $A^{(t)}(t = 2, \ldots, N)$, defined below, represent the equivalent linear system for which the variable x_{t-1} has just been eliminated.

$$A^{(t)} = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & \cdots & a_{1,t-1}^{(1)} & a_{1,t}^{(1)} & \cdots & a_{1,N}^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & \cdots & a_{2,t-1}^{(2)} & a_{2,t}^{(2)} & \cdots & a_{2N}^{(2)} \\ : & : & : & : & : & : & : & : \\ : & & : & : & a_{t-1,t-1}^{(t-1)} & a_{t-1,1}^{(t-1)} & \cdots & a_{t-1,N}^{(t-1)} \\ : & & & : & 0 & a_{t,t}^{(t)} & \cdots & a_{t,N}^{(t)} \\ : & & & & : & : & \cdots & : \\ 0 & \cdots & \cdots & \cdots & 0 & a_{N,t}^{(t)} & \cdots & a_{N,N}^{(t)} \end{bmatrix}.$$

This triangulization procedure will fail if any of the pivot elements is zero. In practice it is often desirable to perform partial or complete pivoting to ensure numerical stability on finite-digit arithmetic. The amount of computation time required by any serial setting of Gaussian elimination to solve a linear system of N equations is proportional to N^3. As for most matrix computations, parallelizing Gaussian elimination method requires a mapping of the matrix elements to the set of processors that allows efficient communication across rows and columns. The most natural interconnection topology that meets this requirement is the mesh. The mesh topology allows to pipeline GE steps where arithmetic computations are overlapped with communication. Another way of parallelizing the GE steps is to use broadcasting to exchange the pivot rows and multipliers columns. Such an approach has been extensively investigated on the hypercube [2,4,10,13]. The broadcasting approach on the hypercube has introduced little improvement over the pipelining approach and it proved inefficient when pivoting is required [4,13].

This paper presents matrix mapping techniques and a parallel implementation of matrix triangulization based on the broadcasting approach for both partial and complete pivoting on the k-ary n-cube. The proposed algorithm is then compared against the existing implementations of the hypercube and the mesh topologies.

The rest of the paper is organized as follows: In Section 2, some related work is presented. In Section 3, we give an overview of the k-ary n-cube. In Section 4, we discuss the employed matrix mapping techniques. Section 5 presents a parallel algorithm for matrix triangulization on the k-ary n-cube. The proposed algorithms are analyzed in Section 6. Conclusions are given in Section 7.

2. Overview of the related work. One of the most widely used methods for solving linear systems is Gaussian elimination. As any other direct method of solution, Gaussian elimination is of $O(N^3)$ serial complexity. Therefore, the implementation of the matrix triangulization in parallel computers has been extensively studied and used by many computer manufacturers as a benchmarking algorithm [15].

The literature contains a vast body of research on parallel linear algebra. Here are some recent examples: Block LU decomposition on the BBN TC200 [14], hypercube based parallel LU factorization [2,4,10,13], matrix triangulization on mesh-connected architectures [5], parallel block LU factorization on the IBM 3090 VF/600J supercomputer [7], and pipelined ring algorithms on a ring of transputers [12]. 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 the finest case each processor holds a single matrix element and in each triangulization step all processors update the corresponding matrix element with just one multiplication, one division, and one subtraction. The total computation time required to triangulize the matrix is approximately $3N$, using N^2 processors.

The major problem with parallel matrix factorization algorithms is that they suffer high communication delays. Pipelined ring algorithms [12] alleviate the communication penalty by overlapping computation and communication. The columns of the matrix are stored in local caches so that partial pivoting requires no communication overhead. Mesh implementations of the pipelined algorithms seem to be more attractive when pivoting is not required. The hypercube broadcasting-based matrix triangulization algorithms could not introduce significant improvement over pipelining [2,4,10,13]. Here we take advantage of the increased connectivity of the k-ary n-cube to improve the communication time requirement of the parallel matrix triangulization. A parallel algorithm based on the broadcasting approach for the k-ary n-cube will be discussed. First we review the k-ary n-cube and then we discuss the employed mapping techniques and present the parallel algorithm for matrix triangulization.

3. The *k*-ary *n*-cube. The k-ary n-cube, $Q_{k,n}$, consists of k^n vertices labeled with strings of n symbols from the set of symbols $\langle k \rangle = \{0, 1, \ldots, k-1\}$. Two nodes $U = u_n u_{n-1} \cdots u_1$ and $V = v_n v_{n-1} \cdots v_1$ are connected if, and only if, $u_j \neq v_j$ for some j, $1 \leq j \leq n$, and $u_i = v_i$ for all i, $1 \leq i \leq n$ and $i \neq j$. In $Q_{k,n}$ a node has degree $n(k-1)$, diameter n, and an average diameter $n(k-1)/k$ [1,9]. Figure 1 shows $Q_{3,2}$.

PROPOSITION 1. *The $Q_{k,n}$ can be partitioned into k^{n-m} disjoint $Q_{k,m}$ subcubes. Each $Q_{k,m}$ subcube can be uniquely represented by a string of $n-m$ symbols from $\langle k \rangle$ followed by m dont'care symbols "*". Denote these*

subcubes by $Q^1_{k,m}, Q^2_{k,m}, \ldots, Q^{k^{n-m}}_{k,m}$. *Alternatively, each* $Q_{k,m}$ *subcube can be uniquely represented by a string of* m *dont'care symbols followed by* $n-m$ *symbols from* $\langle k \rangle$. *Denote these subcubes by* ${}^1Q_{k,m}, {}^2Q_{k,m}, \ldots, {}^{k^{n-m}}Q_{k,m}$.

Figures 2(a) and 2(b) show two different partitionings for the $Q_{3,2}$ cube $\{Q^1_{3,1} \equiv 0^*, Q^2_{3,1} \equiv 1^*, Q^3_{3,1} \equiv 2^*\}$ and $\{{}^1Q_{3,1} \equiv {}^*0, {}^2Q_{3,1} \equiv {}^*1, {}^3Q_{3,1} \equiv {}^*2\}$. We should notice that there are many other ways to partition the k-ary n-cube into disjoint subcubes. In fact, there are $\binom{n}{m}$ different partitionings which depend on where we locate the m dont'care symbols in the n-digit addresses.

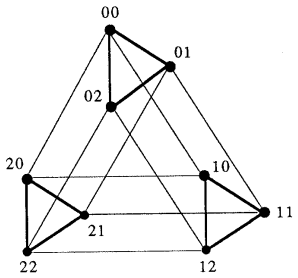
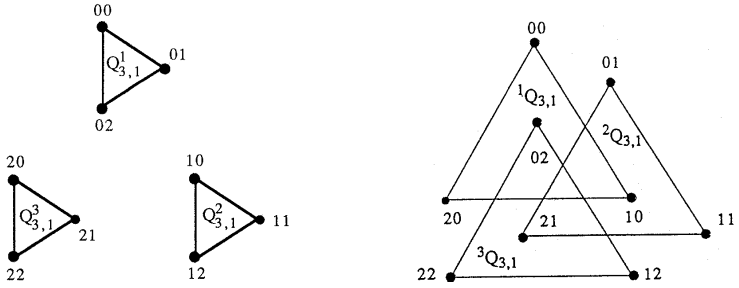


FIG. 1. *The 3-ary 2-cube.*



(a) $Q^1_{3,1} \equiv 0^*$, $Q^2_{3,1} \equiv 1^*$ and $Q^3_{3,1} \equiv 2^*$

(b) ${}^1Q_{3,1} \equiv {}^*0$, ${}^2Q_{3,1} \equiv {}^*1$ and ${}^3Q_{3,1} \equiv {}^*2$

FIG. 2. *Two different partitionings of the 3-ary 2-cube.*

DEFINITION 2. *Let* $\mathbf{I}_n$ *be a one-to-one mapping from the set of strings* $\{u_n u_{n-1} \cdots u_1 | u_i \in \langle k \rangle$ *and* $1 \le i \le n\}$ *onto the set of integers* $\{1, 2, \ldots, k^n\}$. *For any string* $U = u_n u_{n-1} \cdots u_1$ *a unique integer can be generated using the following recursive function:*

$$\mathbf{I}_n(U) = \begin{cases} 1 & \text{if } n = 0 \\ \mathbf{I}_{n-1}(U) + u_n * k^{n-1} & \text{otherwise}. \end{cases}$$

PROPOSITION 3. *Given any two nodes* $U = u_n u_{n-1} \cdots u_1$ *and* $V = v_n v_{n-1} \cdots v_1$ *such that* $\mathbf{I}_{n-m}(u_n u_{n-1} \cdots u_{m+1}) = \mathbf{I}_{n-m}(v_n v_{n-1} \cdots v_{m+1})$, $m \leq n$, *then* U *and* V *belong to the same subcube* $Q_{k,m}$. *Similarly, if* $\mathbf{I}_m(u_m u_{m-1} \cdots u_1) = \mathbf{I}_m(v_m v_{m-1} \cdots v_1)$, *then* U *and* V *belong to the same subcube* $Q_{k,n-m}$.

4. Subcube matrix mapping techniques. In this section we will present subcube matrix mapping techniques that facilitate simultaneous broadcasts across rows and columns. These mapping techniques partition the n-digit addresses of the k-ary n-cube into two parts one with m digits and another with $n - m$ digits. This will necessarily partition the k-ary n-cube into k^m (resp. k^{n-m}) disjoint subcubes of dimension $n - m$ (resp. m). Thus, matrix elements can be distributed over these disjoint subcubes such that elements in the same row (resp. column) reside in the same subcube.

Let $A = \{a_{ij} | 1 \leq i, j \leq N\}$ be a matrix of order N, $A_r = \{r_1, r_2, \ldots, r_N\}$ be the set of rows in the matrix A, $A_c = \{c_1, c_2, \ldots, c_N\}$ be the set of columns in the matrix A, H be the set of nodes in $Q_{k,n}$, and $\rho_r = \{{}^1Q_{k,n-m}, {}^2Q_{k,n-m}, \ldots, {}^{k^m}Q_{k,n-m}\}$ and $\rho_c = \{Q^1_{k,m}, Q^2_{k,m}, \ldots, Q^{k^{n-m}}_{k,m}\}$ be two partitionings of the $Q_{k,n}$, where $N \geq \sqrt{k^n}$ and $m \leq n$.

DEFINITION 4. *The row mapping function* $f_{\text{row}} : A_r \to \rho_r$ *that maps the matrix rows to subcubes on a cyclic manner is defined as follows:* $f(r_i) = {}^R Q_{k,n-m}$ *such that* $R = [(i-1) \bmod k^m] + 1$.

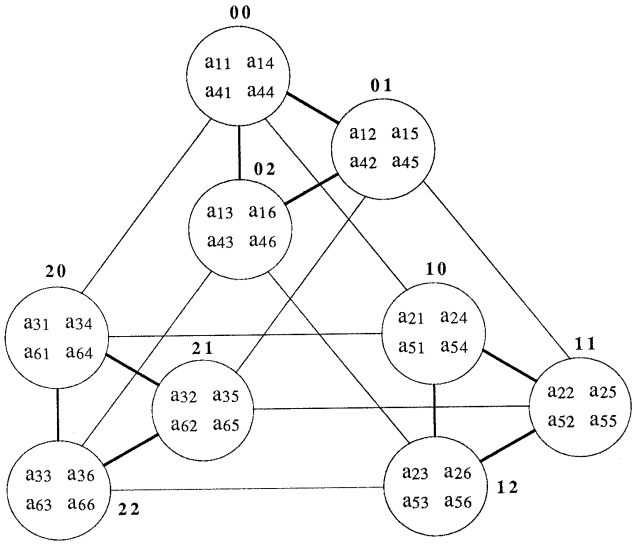


FIG. 3. *Subcube matrix mapping of* $A_{6\times 6}$ *on* $Q_{3,2}$.

DEFINITION 5. *The column mapping function* $f_{\text{col}} : A_c \rightarrow \rho_c$ *that maps the matrix columns to subcubes on a cyclic manner is defined as follows:* $f(c_j) = Q^C_{k,m}$ *such that* $C = [j-1)$ **mod** $k^{n-m}] + 1$.

DEFINITION 6. *The subcube matrix mapping function* $f : A \rightarrow H$ *is defined as follows:* $f(a_{ij}) = U$ *such that* $\mathbf{I}_{n-m}(u_n u_{n-1} \cdots u_{m+1}) = R$ *and* $\mathbf{I}_m(u_m u_{m-1} \cdots u_1) = C$, *where* $R = [(i-1)$ **mod** $k^m] + 1$ *and* $C = [j-1)$ **mod** $k^{n-m}] + 1$. *The node* U *is denoted by* P_{RC}.

The function f distributes the matrix rows over the subcubes cyclically. Within each subcube the row elements are also distributed cyclically over its nodes. Formally speaking, for each R and C, $1 \leq R \leq k^m$ and $1 \leq C \leq k^{n-m}$, let λ_R be the largest integer such that $R + \lambda_R k^m \leq N$ and let μ_C be the largest integer such that $C + \mu_C k^{n-m} \leq N$. A node P_{RC} will be assigned the submatrix $A_{N/k^m, N/k^{n-m}} = [a_{ij}]$, $i = R, R+k^m, R+2k^m, \ldots, R+\lambda_R k^m$ and $j = C, C+k^{n-m}, C+2k^{n-m}, \ldots, C+\mu_C k^{n-m}$. This type of mapping is called cyclic data distribution and is known to achieve good load balancing and processor utilization [11]. Figure 3 shows an example on mapping a 6×6 matrix onto a $Q_{3,2}$ using the mapping function f.

5. Parallel matrix triangulization. Gaussian elimination can be used to triangulize a dense matrix A of order N by generating $N-1$ matrices $A^{(2)}, \ldots, A^{(N)}$ as defined in the introduction section. The steps involved in computing these matrices are given below:

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

Task $\langle T_{tt} \rangle \equiv$ $\begin{cases} \textit{Partial pivoting: interchange the rows of } A^{(t)}, \\ \text{i.e., } |a^{(t)}_{tt}| = \max\limits_{t \leq i \leq N} \{|a^{(t)}_{it}|\} \\ \\ \textit{Complete pivoting: interchange rows \& columns of } A^{(t)}, \\ \text{i.e., } |a^{(t)}_{tt}| = \max\limits_{t \leq i,j \leq N} \{|a^{(t)}_{ij}|\} \end{cases}$

  for $i = t+1$ to N do

    Task $\langle T_{it} \rangle \equiv \{\textit{Compute multipliers}\}\ a^{(t)}_{it} = a^{(t)}_{it} / a^{(t)}_{tt}$

  endfor

  for $i = t+1$ to N do

    for $j = t+1$ to N do

      Task $\langle T^{(t)}_{ij} \rangle \equiv \{\textit{Eliminate}\} a^{(t)}_{ij} = a^{(t)}_{ij} - a^{(t)}_{it} * a^{(t)}_{tj}$

    endfor

  endfor

endfor

In the $Q_{k,n}$ based parallel matrix triangulization each processor will hold one or more matrix elements. Without pivoting processors can compute multipliers $\langle T_{it} \rangle$ and perform eliminations $\langle T^{(t)}_{ij} \rangle$ 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 tth column should find $a_{rt}^{(t)}$ such that $|a_{rt}^{(t)}| = \max_{t \le i \le N}\{|a_{it}^{(t)}|\}$, and then the set of processors holding elements of the tth and the rth rows should interchange the relevant subrows.

To perform complete pivoting the set of processors holding elements of the submatrix $a_{ij}^{(t)}, t \le i, j \le N$, find $a_{rs}^{(t)}$ such that $|a_{rs}^{(t)}| = \max_{t \le i,j \le N}\{|a_{ij}^{(t)}|\}$, then the set of processors holding elements of the tth and the rth rows should interchange the relevant subrows and the set of processors holding elements of the sth and the rth columns should interchange the relevant subcolumns.

The whole process requires $N-1$ steps to triangulize the matrix. At the tth step, processors tasks can be described as follows:

— Processor $P_{R_t,C}$ where $R_t = [(t-1)\ \mathbf{mod}\ k^m] + 1$ and $1 \le C \le k^{n-m}$, should broadcast the pivot subrow $\{a_{tj} | j = C, C + k^{n-m}, C + 2k^{n-m}, \ldots, C + \mu_C k^{n-m}\}$ in $Q_{k,m}^C$.

— Processor P_{R,C_t} where $1 \le R \le k^m$ and $C_t = [(t-1)\ \mathbf{mod}\ k^{n-m}] + 1$, should compute the multipliers subcolumn $\langle T_{it} \rangle$, $i = R, R + k^m, R + 2k^m, \ldots, R + \lambda_R k^m$, and broadcast it in ${}^R Q_{k,n-m}$.

— Processor $P_{R,C}$, where $1 \le R \le k^m$ and $1 \le C \le k^{n-m}$ excluding those holding the multipliers subcolumns should wait until multipliers subcolumn are received then eliminate the submatrix $\{a_{ij} | i = R, R + k^m, R + 2k^m, \ldots, R + \lambda_R k^m, j = C, C + k^{n-m}, C + 2k^{n-m}, \ldots, C + \mu_C k^{n-m}$, and $t \le i, j \le N\}$. Processors holding the multipliers subcolumns should not wait and may proceed with elimination.

The parallel matrix triangulization program that is executed in each node $U = (u_n u_{n-1} \cdots u_1)$ in $Q_{k,n}$ given in Fig. 4. The program assumes that the matrix is initially distributed using the mapping function f. The procedure *broadcast* performs broadcasting in a subcube. Optimal broadcasting on a k-ary n-cube would be of $O(n)$ complexity. When a processor executes the procedure *wait* it keeps checking all ports until the specified value is received.

For partial pivoting the tth pivot row is determined by the set of processors $\{P_{R,C_t} | 1 \le R \le k^m$ and $C_t = [(t-1)\ \mathbf{mod}\ k^{n-m}] + 1\}$. 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 *imax.* If the rows t and *imax* are in ${}^{R_t} Q_{k,n-m}$ where $R_t = [(t-1)\ \mathbf{mod}\ k^m] + 1$ then each processor in ${}^{R_t} Q_{k,n-m}$ should swap subrows a_{ij} and $a_{imax,j}$, $j = C, C + k^{n-m}, C + 2k^{n-m}, \ldots, C + \mu_C k^{n-m}$. Otherwise, each processor pair $P_{R_t,C}$ and $P_{v_t,C}$ where $V_t = [(imax - 1)\ \mathbf{mod}\ k^m] + 1$ and $1 \le C \le k^{n-m}$, should exchange subrows a_{ij} and $a_{imax,j}$, $j = C, C + k^{n-m}, C + 2k^{n-m}, \ldots, C + \mu_C k^{n-m}$. The task $\langle T_{tt} \rangle$ in partial pivoting is described in Fig. 5.

The two procedures *Swap_subrows* and *Interchange_subrows* are used to swap rows within the same processor and between two different processors in a subcube, 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. It then issues an asynchronous receive to get the subrow i from the destination processor followed by a synchronous send to send the subrow j to the same destination processor and then blocks until the asynchronous receive is complete.

For complete pivoting all processors in the $Q_{k,n}$ will perform "exchange-max" to find *imax* and *jmax* such that $|a_{imax,jmax}| = \max\{|a_{ij}| t \leq i, j \leq N\}$. Then, the tth pivot row is determined by swapping the rows t and *imax* and swapping the columns t and *jmax.* This is achieved as follows: if the rows t and *imax* are in the same subcube ${}^{R_t}Q_{k,n-m}$ where $R_t = [(t-1) \textbf{ mod } k^m] + 1$, then each processor in ${}^{R_t}Q_{k,n-m}$ should swap subrows a_{tj} and $a_{imax,j}$, $j = C$, $C + k^{n-m}$, $C + 2k^{n-m}, \ldots, C + \mu_C k^{n-m}$. If the rows t and *imax* are in two different subcubes ${}^{R_t}Q_{k,n-m}$ and ${}^{V_t}Q_{k,n-m}$ where $V_t = [(imax - 1) \textbf{ mod } k^m] + 1$, then each processor pair $P_{R_t,C}$ and $P_{V_t,C}$ where $1 \leq C \leq k^{n-m}$ should interchange subrows a_{tj} and $a_{imax,j}$, $j = C$, $C+k^{n-m}$, $C+2k^{n-m}, \ldots, C+\mu_C k^{n-m}$. Similar processing is to be performed

```
Program Parallel_Matrix_Triangulization
    { the following code is executed by the node U = u_n u_{n-1} ... u_1 in Q_{k,n} }
    compute R = I_{n-m}(u_n u_{n-1} ... u_{m+1}) and C = I_m(u_m u_{m-1} ... u_1)
    for t = 1 to N - 1 do
        R_t = [(t - 1) mod k^m] + 1
        C_t = [(t - 1) mod k^{n-m}] + 1
        execute <T_tt>
        if R = R_t then broadcast ({a_tj | j = C, C + k^{n-m}, C + 2k^{n-m}, ...,
            C + mu_C k^{n-m} and (t <= j <= N)}) in Q^C_{k,m}
        if R != R_t then wait ({a_tj | j = C, C + k^{n-m}, C + 2k^{n-m}, ...,
            C + mu_C k^{n-m} and (t <= j <= N)})
        if C = C_t then
            execute <T_it>, i = R, R + k^m, R + 2k^m, ..., R + lambda_R k^m
                and t < i <= N
            broadcast ({a_it | i = R, R + k^m, R + 2k^m, ..., R + lambda_R k^m
                and t < i <= N}) in {}^R Q_{k,n-m}
        endif
        if (C != C_t) then wait ({a_it | i = R, R + k^m, R + 2k^m, ..., R + lambda_R k^m
            and t < i <= N})
        execute <T^(t)_ij>, i = R, R + k^m, R + 2k^m, ..., R + lambda_R k^m, j = C,
            C + k^{n-m}, C + 2k^{n-m}, ..., C + mu_C k^{n-m}, t < i <= N,
            and t <= j <= N
    endfor
end Parallel_Matrix_Triangulization
```

FIG. 4. *Parallel matrix triangulization algorithm on the k-ary n-cube.*

Task $\langle T_{tt} \rangle$: Partial pivoting
    if $C = C_t$ then
        let $M_1 = -1$ and $imax_1 = 1$
        find $imax_1$ such that $|a_{imax_{1,t}}| = \max\{|a_{i,t}| \; | i = R, R + k^m,$
            $R + 2k^m, \ldots, R + \lambda_R k^m$ and $t < i \leq N\}$
        if $(t \leq imax_1 \leq N)$ then let $M_1 = |a_{imax_{1,t}}|$
        for $j = 1$ to m do
            asynchronously receive $(M_i, imax_i)$, $i = 2, \ldots, m(k-1)+1$
                from all neighbors in $Q^C_{k,m}$
            synchronously send $(M_1, imax_1)$ to all neighbors in $Q^C_{k,m}$
            wait for asynchronous receive to complete
            find h such that $M_h = \max\{M_i | i = 1, 2, \ldots, m(k-1)+1\}$
            let $M_1 = M_h$ and $imax_1 = imax_h$
        endfor
        broadcast $(imax_1)$ in ${}^R Q_{k,n-m}$
    else wait $(imax_1)$
    $V_t = ((imax_1 - 1) \bmod k^m) + 1$
    if $R = V_t = R_t$ then Swap_subrows $(t, imax_1)$
    elseif $R = V_t \neq R_t$ then Interchange_subrows $(t, imax_1, R_t)$
    elseif $R = R_t \neq V_t$ then Interchange_subrows $(imax_1, t, V_t)$
    endif
end Task $\langle T_{tt} \rangle$

FIG. 5. *Parallel partial pivoting on the k-ary n-cube.*

for columns t and $jmax$. The k-ary n-cube based complete pivoting algorithm is given in Fig. 6.

To find $imax$ and $jmax$ such that $|a_{imax,jmax}| = \max\{|a_{ij}| t \leq i, j \leq N\}$ all processors in $Q_{k,n}$ execute exchange-max procedure simultaneously. This procedure iteratively enters 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. The binary n-cube based *exchange-max* procedure requires n 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.

Whether single or multiple port communication is used will not affect the performance of the exchange-max procedure for the binary n-cube. This is due to the fact that even with simultaneous send/receive the maximum value requires n steps to travel from an arbitrary node to the farthest node in the graph. However, simultaneous send/receive operations significantly improve

Task $\langle T_{tt} \rangle$: Complete pivoting
  let $M_1 = -1$, $imax_1 = 1$ and $jmax_1 = 1$
  find $imax_1$ and $jmax_1$ such that
  $|a_{imax_1,jmax_1}| = \max\{|a_{i,j}| \; |i = R,\, R + k^m,\, R + 2k^m, \ldots,\, R + \lambda_R k^m,$
    $j = C,\, C + k^{n-m},\, C + 2k^{n-m}, \ldots,\, C + \mu_C k^{n-m}$ and $t \le i,\, j \le N\}$
  if $(t \le imax_1\,, jmax_1 \le N)$ then let $M_1 = |a_{imax_1,jmax_1}|$
  for $j = 1$ to n do
    asynchronously receive $(M_i,\, imax_i,\, jmax_i)$, $i = 2, \ldots,$
      $n(k-1)+1$ from all neighbors
    synchronously send $(M_1,\, imax_1,\, jmax_1)$ to all neighbors
    wait for asynchronous receive to complete
    find h such that $M_h = \max\{M_i | i = 1,\, 2, \ldots,\, n(k-1)+1\}$
    let $M_1 = M_h$, $imax_1 = imax_h$ and $jmax_1 = jmax_h$
  endfor
  $RV_t = ((imax_1 - 1) \bmod k^m) + 1$
  $CV_t = ((jmax_1 - 1) \bmod k^{n-m}) + 1$
  if $(RV_t = R_t)$ and $(CV_t = C_t)$ then
    if $R = R_t$ then *Swap_subrows* $(t,\, imax_1)$
    if $C = C_t$ then *Swap_subcolumns* $(t,\, jmax_1)$
  elseif $(RV_t = R_t)$ and $(CV_t \ne C_t)$ then
    if $R = R_t$ then *Swap_subrows* $(t, imax_1)$
    if $C = CV_t$ then *Interchange_subcolumns* $(t,\, jmax_1,\, C_t)$
    if $C = C_t$ then *Interchange_subcolumns* $(jmax_1,\, t,\, CV_t)$
  elseif $(RV_t \ne R_t)$ and $(CV_t = C_t)$ then
    if $C = C_t$ then *Swap_subcolumns* $(t,\, jmax_1)$
    if $R = RV_t$ then *Interchange_subrows* $(t,\, imax_1,\, R_t)$
    if $R = R_t$ then *Interchange_subrows* $(imax_1,\, t,\, RV_t)$
  elseif $(RV_t \ne R_t)$ and $(CV_t \ne C_t)$ then
    if $R = RV_t$ then *Interchange_subrows* $(t,\, imax_1,\, R_t)$
    if $R = R_t$ then *Interchange_subrows* $(imax_1,\, t,\, RV_t)$
    if $C = CV_t$ then *Interchange_subcolumns* $(t,\, jmax_1,\, C_t)$
    if $C = C_t$ then *Interchange_subcolumns* $(jmax_1,\, t,\, CV_t)$
  endif
end Task $\langle T_{tt} \rangle$

FIG. 6. *Complete pivoting on the k-ary n-cube.*

the performance in the $Q_{k,n}$. What is important is how much time it takes to carry out a single iteration in an $Q_{k,n}$ based exchange-max procedure. In each iteration a processor receives $n(k-1)$ values to be compared with the local extreme. In the worst case, this is done serially which will result in $O(kn^2)$ complexity of the exchange-max procedure. In the best case these $n(k-1)$ values can be compared in $\log n(k-1)$ steps using a system function similar to sum-reduction supported by most of parallel computers. Thus, the $Q_{k,n}$ exchange-max procedure requires $O(n \log n(k-1))$ time to find the maximum.

6. Performance analysis. In this section we present a model for timing of computation and interprocessor communication for the proposed algorithms. Also we compare the performance of the proposed algorithms with those of the hypercube and the mesh networks.

The k-ary n-cube based matrix triangulization algorithm requires $N-1$ serial steps to triangulize a matrix A of order N. In the tth step the following tasks are performed in sequence: (1) locating the pivot row $\langle T_{tt}\rangle$, (2) simultaneous sub-row broadcasts in subcubes of dimension m, (3) N/k^m sequential $\langle T_{it}\rangle$, (4) simultaneous sub-column broadcasts in subcubes of dimension $n-m$, and (5) N^2/k^n sequential $\langle T_{ij}^{(t)}\rangle$. Thus, the execution time T for the parallel matrix triangulization algorithm is given by the following model:

$$T = (N-1)\left(\mathcal{T}_{\langle T_{tt}\rangle} + \mathcal{T}_R + \frac{N}{k^m}\mathcal{T}_{\langle T_{it}\rangle} + \mathcal{T}_C + \frac{N^2}{k^n}\mathcal{T}^{(t)}_{\langle T_{ij}\rangle}\right)$$

where $\mathcal{T}_{\langle T_{tt}\rangle}$ is the total time required to perform $\langle T_{tt}\rangle$ which is equal to the time needed to locate the pivot row plus the time needed to route subrows/subcolumns, $\mathcal{T}_R$ is the time required to broadcast a pivot subrow in $Q_{k,m}$, $\mathcal{T}_C$ is the time required to broadcast a multiplier subcolumn in $Q_{k,n-m}$, $T_{\langle T_{it}\rangle}$ is the time required to perform $\langle T_{it}\rangle$, and $\mathcal{T}_{\langle T_{ij}^{(t)}\rangle}$ is the time required to perform $\langle T_{ij}^{(t)}\rangle$.

The communication time required for broadcasting a message M in a graph with a diameter δ is commonly described by $\delta(t_s + \alpha|M|)$ where t_s is the communication startup, α is the channel throughput, and $|M|$ is the message size [2,4,6]. Applying this model to our primitives we obtain:

$$\mathcal{T}_R = m(t_s + \alpha(N/k^{n-m}))$$

$$\mathcal{T}_C = (n-m)(t_s + \alpha(N/k^m))$$

$$\mathcal{T}_{\langle T_{tt}\rangle} = \mathcal{T}_{\text{emax}} + \mathcal{T}_{\text{route}}$$

$$\mathcal{T}_{\text{emax}} = \begin{cases} t_c \log(N/k^m) + m(ts + 2\alpha + t_c \log(m(k-1))) & \text{partial pivoting} \\ t_c \log(N^2/k^n) + n(t_s + 3\alpha + t_c \log(n(k-1))) & \text{complete pivoting} \end{cases}$$

$$\mathcal{T}_{\text{route}} = \begin{cases} m(t_s + \alpha(N/k^{n-m})) & \text{partial pivoting} \\ m(t_s + \alpha(N/k^{n-m})) + (n-m)(t_s + \alpha(N/k^m)) & \text{complete pivoting} \end{cases}$$

$$\mathcal{T}_{\langle T_{it}\rangle} = t_{fpo}$$

$$\mathcal{T}_{\langle T_{ij}^{(t)}\rangle} = 2t_{fpo}$$

where t_c is the time required to compare two floating-point numbers and t_{fpo} is the average time required to perform a floating-point operation. The above

estimates of execution time represent the worst case analysis of the k-ary n-cube based matrix triangulization algorithm. Furthermore, overlapping of communication and computation is ignored. With these assumptions, the models for estimating computation time with partial and complete pivoting, $\tau^{(p)}_{\text{comp}}$ and $\tau^{(c)}_{\text{comp}}$, and for estimating communication time with partial and complete pivoting, $\tau^{(p)}_{\text{comm}}$ and $\tau^{(c)}_{\text{comm}}$, are given below.

$$\tau^{(p)}_{\text{comp}} = (N-1)\left(\mathbf{t}_{fpo}\frac{k^{n-m}N+2N^2}{k^n} + \mathbf{t}_c\left(\log\frac{N}{k^m} + m\log(m(k-1))\right)\right)$$

$$\tau^{(c)}_{\text{comp}} = (N-1)\left(\mathbf{t}_{fpo}\frac{k^{n-m}N+2N^2}{k^n} + \mathbf{t}_c\left(\log\frac{N^2}{k^n} + n\log(n(k-1))\right)\right)$$

$$\tau^{(p)}_{\text{comm}} = (N-1)\left(\mathbf{t}_s(n+2m) + \alpha\left(\frac{2Nmk^m + N(n-m)k^{n-m} + 2mk^n}{k^n}\right)\right)$$

$$\tau^{(c)}_{\text{comm}} = (N-1)\left(\mathbf{t}_s(3n) + \alpha\left(\frac{2Nmk^m + 2N(n-m)k^{n-m} + 3nk^n}{k^n}\right)\right).$$

The above expressions indicate that the proposed parallel algorithm requires $O(N^3/k^n)$ computation complexity to triangulize a matrix of order N on a k-ary n-cube. This is the same computation time requirement for an optimal hypercube or mesh implementation on graphs of similar sizes. However, the communication time requirement of a k-ary n-cube based implementation is much better than the hypercube and the mesh as discussed below.

The cost of broadcasting in a graph is generally measured by the required number of broadcasting steps regardless of the dimension of the message being broadcasted. This is due to the fact that today's networking technology offers higher bandwidth that can deliver up to several giga bits transmission rates. Hence, what actually matters is the initial transmission startup time. This will be the major criterion in comparing the communication time requirement of the k-ary n-cube based matrix triangulization, the hypercube based implementation, and the mesh based implementation.

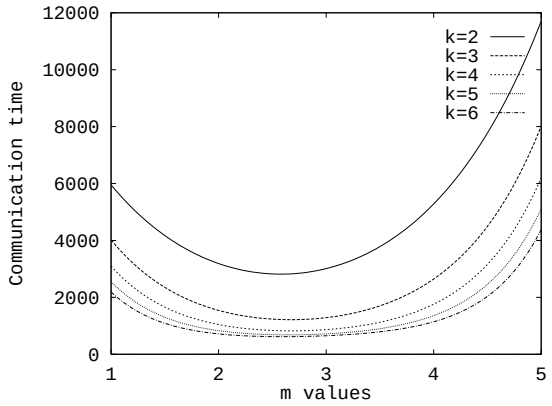
The optimal number of broadcasting steps in any parallel setting of the matrix triangulization problem on the mesh can be achieved using a square mesh, say $M(\sqrt{k^n}, \sqrt{k^n})$ mesh with k^n nodes. In this setting $O(N\sqrt{k^n})$ broadcasting steps would be required to broadcast the pivot rows and multipliers columns in an $M(\sqrt{k^n}, \sqrt{k^n})$ mesh.

The smallest hypercube of size k^n nodes is $Q_{\lceil n\log k\rceil}$. An obvious subcube partitioning that achieves the best broadcasting results is to divide the $\lceil n\log k\rceil$ bits of the $Q_{\lceil n\log k\rceil}$ into two equal parts. This will result in $Q_{\frac{\lceil n\log k\rceil}{2}}$ broadcasts in order to do the required pivot rows and multipliers

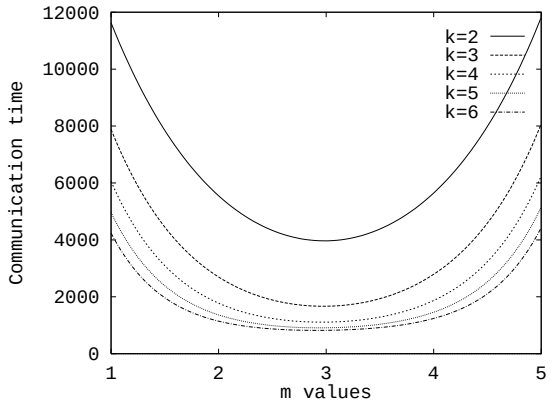
columns routings. Therefore, an optimal hypercube implementation would end up $O(Nn \log k)$ communication time requirements [2,4].

The above expressions for $\tau_{\text{comm}}^{(p)}$ and $\tau_{\text{comm}}^{(c)}$ indicate that the k-ary n-cube based matrix triangulization uses $O(Nn)$ communication time to triangulize a matrix of order N on the k-ary n-cube. This compares favorably to the hypercube and the mesh communication time requirements.

Figure 7 shows the communication time requirements of the k-ary n-cube based matrix triangulization for different k values. Here we choose $\alpha = 1$ and $t_s = 1000$ μs. These choices are representative of the currently available machines [8]. The optimal communication time is achieved at $m = n/2$ in both partial and complete pivoting. In partial pivoting the communication cost increases faster for $m > n/2$ than for $m < n/2$. This is due to the fact that partial pivoting does not require row communication in order to locate



(a) *Partial pivoting.*



(b) *Complete pivoting.*

FIG. 7. *Communication cost of the k-ary n-cube matrix triangulization algorithm.*

the pivot rows (recall that each processor will be assigned a submatrix of dimensions $N/k^m \times N/k^{n-m}$).

The communication cost improves with larger k values in both partial and complete pivoting. This improvement comes at the expense of higher network cost. To give better indication of the efficiency of the algorithm we should take into consideration the network cost. One reasonable measure for the network cost is the total number of edges required to realize the network. This measure captures both the cost of the links and the cost of the I/O ports of all nodes. For instance, in the k-ary n-cube there are $(n(k-1)k^n)/2$ links and $n(k-1)k^n$ I/O ports, while a hypercube of similar size has $(n \log k\, k^n)/2$ links and $(n \log k\, k^n)$ I/O ports. Hence, the performance improvement at a factor of $\log k$ comes at the expense of increasing the network cost by a factor of $(k-1)/\log k$. This is demonstrated more clearly in Fig. 8. The figure shows the cost/performance ratio of the proposed algorithm. In this figure the parameters n and N (the cube dimension and the matrix order) are fixed. The cost/performance ratio grows faster for higher radix values on both partial and complete pivoting. The information in Figs. 7 and 8 indicate that the extra connectivity of the higher radix cubes contribute little towards reducing the communication cost. Therefore, a cost-performance trade-off would be a practical solution.

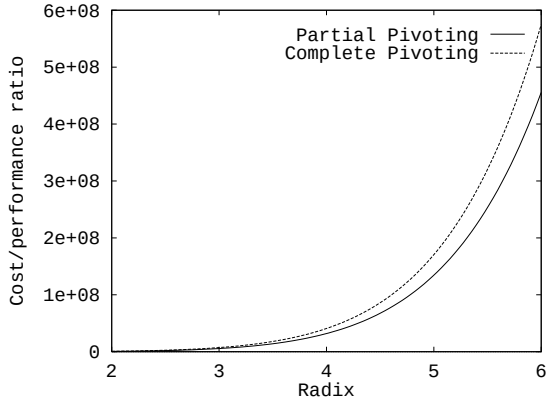


FIG. 8. *Cost/performance ratio of the k-ary n-cube matrix triangulization algorithm.*

7. Conclusion. The major contribution of this paper is the design and evaluation of a parallel matrix triangulization algorithm based on the k-ary n-cube topology. Parallel algorithms for carrying out partial and complete pivoting are also discussed. Compared to the hypercube and the mesh based matrix triangulization [2,4,5,10,13], the k-ary n-cube based parallel matrix triangulization 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 k-ary n-cube than the hypercube and the mesh. When communication along all channels can take place simultaneously, an optimal broadcasting in the k-ary n-cube requires $O(n)$ steps which is much lower than $O(n \log k)$ steps that are required by an optimal broadcasting in a hypercube or $O(\sqrt{k^n})$ steps that are required by an optimal broadcasting in a mesh. Second, the exchange of rows/columns can be done more efficiently between the k-ary n-cube processors since the average distance of the k-ary n-cube is less than that of the hypercube and the mesh. Finally, we have presented timing models for estimating computation and communication time of the k-ary n-cube based matrix triangulization for both partial and complete pivoting. Also we have conducted a cost-performance analysis of the proposed algorithm.

REFERENCES

[1] D. Agrawal and L. Bhuyan, "Generalized hypercube and hyperbus structures for a computer network", *IEEE Transactions on Computers* **33**, 323 (1984).

[2] M. Angelacci and M. Colajanni, "Subcube matrix decomposition: A unifying view of LU factorization on multicomputers", *Parallel Computing* **20**, 257 (1994).

[3] R. Burden *et al.*, *Numerical Analysis*, 2nd ed. (PWS Publishers, 1981).

[4] P. R. Cappelo, "Gaussian elimination on hypercube automaton", *Journal of Parallel and Distributed Computing* **4**, 288 (1987).

[5] P. R. Cappelo, "A mesh automaton for solving dense linear systems", *Proceedings of the International Conference on Parallel Processing*, 1985, pp. 418–425.

[6] M. Cosnard, Y. Robert and B. Tourancheau, "Evaluating speedup on distributed memory architecture", *Parallel Computing* **10**, 247 (1989).

[7] K. Dackland *et al*, "Design and evaluation of parallel block algorithms: LU factorization on the IBM 3090 VF/600J", *Proceedings of the Fifth SIAM Conference on Parallel Processing for Scientific Computing* 3 (1992).

[8] S. W. Graham and S. R. Seidel, "The cost of broadcasting on the star graphs and the k-ary hypercubes", *IEEE Transactions on Computers* **42**, 754 (1993).

[9] S. Lakshimvarahan and S. Dhall, "A new hierarchy of hypercube interconnection schemes for parallel computers", *Journal of Supercomputing* **2**, 81 (1988).

[10] G. Laszewski *et al.*, "On the parallelization of blocked LU factorization algorithms on distributed memory architectures", *Supercomputing'92*, 1992, pp. 170–179.

[11] W. Lictenstein and S. L. Johnsson, "Block cyclic dense linear algebra", *SIAM J. Sci. Comp.* **14**, 1257 (1993).

[12] B. V. Purushotham, A. Basu and P. S. Kumar, "Performance estimation of LU factorization of message passing multiprocessors", *Parallel Processing Letters* **2**, 51 (1992).

[13] Y. Saad, "Gaussian elimination on hypercubes", in *Parallel Algorithms and Architectures*, eds. M. Cosnard *et al.* (Elsevier Science, 1986), pp. 5–18.

[14] S. Starts and A. N. Beris, "LU decomposition optimized for parallel computer with a hierarchical distributed memory", *Parallel Computing* **18**, 959 (1992).

[15] B. Tourancheau, "LU factorization on FPST series hypercube", *Parallel and Distributed Computing* (Elsevier Science, 1989).