A New Topology with Odd Degree for Multiprocessor Systems

RAJIB K. DAS AND BHABANI P. SINHA

Electronics Unit, Indian Statistical Institute, 203 B.T. Road, Calcutta 700 035, India

A new topology for interconnection networks has been proposed. The underlying network graph has $N = 4^n$ nodes ($n \geq 2$) and is *almost* regular with maximum degree 5 and diameter $\leq \lfloor 3/4 \log_2 N \rfloor + 1$. Algorithms for point-to-point routing and single node broadcast have also been developed. It has also been shown that various algorithms for real life applications, e.g., matrix transpose, matrix multiplication, finding the sum/average/maximum/minimum of a set of data elements and ASCEND/DESCEND types of algorithms can be efficiently implemented on this topology. Finally, the underlying idea of constructing this network has been generalized to define a family of *almost regular* odd degree graphs of maximum degree $2j + 1$, $(j > 2)$ with $N = (2j)^n$ nodes and diameter $\lfloor 3/4 \log_j N \rfloor + 1$.

1. INTRODUCTION

Interconnection networks play a major role in the design of efficient parallel and distributed computing systems. Various topologies for static interconnection networks have been proposed in the literature.

Among the regular topologies there are two distinct categories. The hypercubes [6], multidimensional meshes and tori, star graphs and pancake graphs [1], etc. constitute one class where the degree of a network increases with an increase in the number of nodes. The other class includes the constant degree networks like ring [4], chordal ring [10], distributed loop network [5], cube-connected cycles [7], Moebius graph [9], de Bruijn graph [2], Kautz graph [8], etc. All the graphs in the latter category, except the first three, have diameters of the order of $O(\log N)$ with N nodes. Fault-tolerances of the 4-regular de Bruijn and Kautz graphs are better than those of 3-regular Moebius graphs and cube-connected cycles.

In this paper we propose a new family of *almost* regular graphs with odd degrees. These graphs will be suitable for multiprocessor systems as they have (i) a constant node degree of 5, (ii) small diameter, and (iii) moderately simple routing algorithms. The connections among the nodes are defined in a functional form which leads to easy and efficient implementations of different classes of algorithms.

The proposed family of network graphs will have $N = 4^n$ nodes, $(n \geq 2)$ with a diameter $\leq \lfloor (3n)/2 \rfloor + 1 = \lfloor 3/4 \log_2 N \rfloor + 1$. For even n, these graphs will be regular of order 5. For odd n, all but 4 vertices will have degree 5 and the remaining 4 vertices will have degree 4. That is, these graphs are regular (for even n) or *almost regular* (for odd n). A heuristic algorithm for point-to-point routing in such *almost* 5-regular graphs has also been suggested. Simulation of the algorithm shows that the length of the path computed by this algorithm does not exceed the shortest distance between them by 1.03, 1.27, and 1.52 on an average for $n = 256$, 1024, and 4096, respectively. An algorithm for single node broadcast has also been presented. This algorithm requires $3n$ time for a graph with 4^n nodes. Implementations of various algorithms for real-life applications, e.g., matrix transpose, matrix multiplication, finding the sum/average/maximum/minimum of a set of data elements, and ASCEND/DESCEND types of algorithms [7], have also been discussed.

Finally we have generalized the underlying ideas of the construction of this topology to define families of such *almost regular* odd degree graphs of maximum degree $2j + 1$, $(j > 2)$ with $N = (2j)^n$ nodes, $(n \geq 2)$ and diameter $\leq \lfloor (3n)/2 \rfloor + 1 = \lfloor 3/4 \log_j N \rfloor$.

2. THE PROPOSED GRAPH AND SOME OF ITS PROPERTIES

We define the proposed graph $G = (V, E)$ with V as the vertex set and E as the set of undirected edges in the following way:

(1) Consider a string $v_1 v_2 \cdots v_i \cdots v_n$, $v_i \in \{0, 1, 2, 3\}$, $1 \leq i \leq n$. Every distinct string of the form $v_1 v_2 \cdots v_i \cdots v_n$ has a corresponding distinct vertex $v \in V$. Clearly, $|V| = 4^n$.

(2) For $v \in V$ and $k \in \{1, 2\}$, we define functions $f_k(v)$, $\varphi_k(v)$ and $g(v)$ as follows:

$$f_k(v_1 v_2 \cdots v_n) = u_1 u_2 \cdots u_n,$$
$$\text{where } u_i = \begin{cases} v_{i+1}, & \text{for } 1 \leq i \leq n-1 \\ (v_1 + k) \bmod 4, & \text{for } i = n \end{cases}$$

$$\varphi_k(v_1 v_2 \cdots v_n) = u_1 u_2 \cdots u_n,$$
$$\text{where } u_i = \begin{cases} v_{i-1} & \text{for } 2 \leq i \leq n \\ (v_n - k) \bmod 4, & \text{for } i = 1 \end{cases}$$

$$g(v_1 v_2 \cdots v_n) = u_1 u_2 \cdots u_n,$$
$$\text{where } u_i = \begin{cases} v_i & \text{for } 1 \leq i \leq n-2 \\ (v_i + 2) \bmod 4, & \text{for } i = n-1, n. \end{cases}$$

Note that $f_k^{-1} = \varphi_k$ and $g^{-1} = g$.

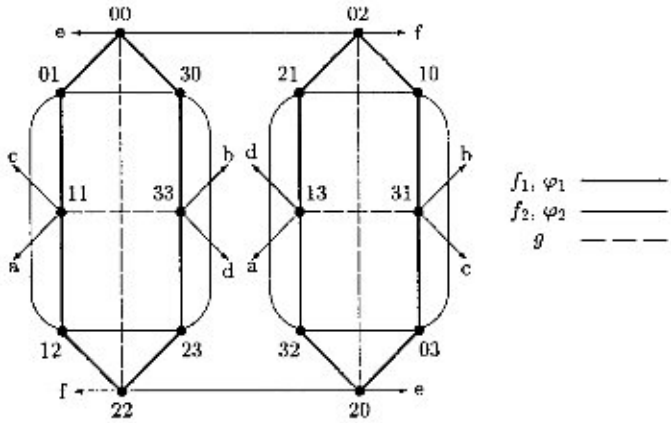

FIG. 1. A graph with 16 nodes.

Now the following steps complete the definition of the graph $G(V, E)$:

(i) Connect an edge from u to v if and only if $f_k(u) = v$ or $\varphi_k(u) = v$ or $g(u) = v$.
(ii) Remove the directions of all the edges.

It is clear that the maximum degree of a vertex $v \in V$ is 5. A graph with 16 nodes is shown in Fig. 1.

In what follows, $v_1 v_2 \cdots v_n$ will denote the string representation of a vertex v. Any digit in a string will be regarded as a modulo 4 number, and the corresponding arithmetic involving such digits will also be in modulo 4, unless otherwise mentioned.

2.1. Degree Distribution

It can be easily shown [3] that, $\forall u \in V$, (i) $f_1(u) \neq f_2(u)$, (ii) $\varphi_1(u) \neq \varphi_2(u)$, (iii) for $i = 1, 2$, $f_i(u) \neq g(u)$ and $\varphi_i(u) \neq g(u)$, (iv) for $i, j \in \{1, 2\}$, $f_i(u) = \varphi_j(u)$, only if $i = j = 2$ and n is odd, with u of the form $u_1(u_1 + 2)u_1 \cdots (u_1 + 2)u_1$, where $u_1 \in \{0, 1, 2, 3\}$. Hence, we get the following lemma:

LEMMA 1. *In $G(V, E)$, with $|V| = 4^n$, (i) all nodes will be of degree 5 when n is even, and (ii) only 4 nodes will be of degree 4 and the rest of degree 5, when n is odd.*

3. PATH LENGTH AND DIAMETER

We now consider a path from a source node $s = s_1 s_2 \cdots s_n$ to a destination node $t = t_1 t_2 \cdots t_n$. Since the links of the graph $G(V, E)$ are characterized by the functions f_i, φ_i, and g, a path of length k can be represented by a string $p_1 p_2 \cdots p_k$, where $p_i \in \{f_1, f_2, \varphi_1, \varphi_2, g\}$. First we consider the paths which do not involve φ_1 or φ_2. Also, as $g^2 =$ Id, the identity function, we would not apply two g's in succession. To investigate the nature of such paths we define the following:

DEFINITION 1. For two digits $a, b \in \{0, 1, 2, 3\}$, we define an operation $\bullet$ as

$$b \bullet a = \begin{cases} 0, & \text{if } (b - a) \bmod 4 = 1 \text{ or } 2 \\ 1, & \text{if } (b - a) \bmod 4 = 3 \text{ or } 0. \end{cases}$$

We consider a path P, starting from the node s and of the form $h_1 g_1 h_2 g_2 \cdots h_i g_i \cdots h_n g_n$, where $h_i \in \{f_1, f_2\}$ and $g_i =$ Id or g. Let the path P lead to a node $u = u_1 u_2 \cdots u_n$. Corresponding to the path P we define two strings $c = c_1 c_2 \cdots c_n$, $c_i \in \{1, 2\}$ and $x = x_1 x_2 \cdots x_n$, $x_i \in \{0, 1\}$ as

(i) if $h_i = f_k$, $k \in \{1, 2\}$, then $c_i = k$
(ii) if $g_i =$ Id, then $x_i = 0$ and if $g_i = g$ then $x_i = 1$.

LEMMA 2. *For $1 \leq i \leq n$, the following relation holds*

$$u_i = \begin{cases} (s_i + c_i), & \text{if } x_i \oplus x_{i+1} = 0 \\ (s_i + c_i) + 2, & \text{if } x_i \oplus x_{i+1} = 1, \end{cases}$$

where $x_{n+1} = x_1$.

Proof. The function $h_i (= f_k)$ operated on a string modifies the first digit, say a, of the string to $a + k$, i.e., $a + c_i$, and puts it in the nth digit position of the resulting string. Since the total number of h_i's in P is same as the total number of digits in s, it follows that $u_i = s_i + c_i$ or $s_i + c_i + 2$. Addition of the term c_i is due to the function f_i and absence or presence of the term 2 is determined by the values of x_i and x_{i+1}, as each g function involves addition by 2. In fact, there can be 4 possible cases.

(i) $x_i = 0$, $x_{i+1} = 0$. No g function is involved. Hence, $u_i = s_i + c_i$.
(ii) $x_i = 0$, $x_{i+1} = 1$. Only one g function is involved and hence $x_i = 1$, $u_i = s_i + c_i + 2$.
(iii) $x_i = 1$, $x_{i+1} = 0$. $u_i = s_i + c_i + 2$.
(iv) $x_i = 1$, $x_{i+1} = 1$. Two g functions are involved and hence $u_i = s_i + c_i + 2 + 2 = s_i + c_i$.

Hence, the proof. ■

Note that $b \bullet a$ can take on only binary values, i.e., 0 or 1. From the result of Lemma 2 we arrive at the following results.

LEMMA 3. *For $1 \leq i \leq n$, $x_i \ominus x_{i+1} = u_i \bullet s_i$, where $x_{n+1} = x_1$.*

Proof. Follows from Lemma 2 and Definition 1. ■

LEMMA 4. *$x_1 = (\Sigma\ (u_i \bullet s_i)) \oplus x_1$, where Σ stands for modulo 2 sum over $i = 1, 2, \ldots, n$.*

Proof. From Lemma 3, $x_i \oplus x_{i+1} = u_i \bullet s_i$ for $1 \leq i \leq n$. We can write this equation as $x_{i+1} = (u_i \bullet s_i) \oplus x_i$, for $i = 1, 2, \ldots, n$, where $x_{n+1} = x_1$.
Solving these equations, we get $x_1 = (\Sigma\,(u_i \bullet s_i)) \oplus x_1$. ■

COROLLARY. *It follows that $\Sigma\ (u_i \bullet s_i) = 0$.*

LEMMA 5. *For any pair of nodes (s, t) such that $\Sigma\ t_i \bullet s_i = 0$, there exists a path P between s and t, where $P = h_1 g_1 h_2 g_2 \cdots h_i g_i \cdots h_n g_n$, $h_i \in \{f_1, f_2\}$.*

Proof. The proof involves finding the strings c and x which will uniquely characterize the path P. By Lemma 2, $t_i = s_i + c_i$ or $s_i + c_i + 2$. Hence, we find c_i by the

following equation:

$$c_i = \begin{cases} 1, & \text{if } t_i - s_i = 1 \text{ or } 3 \\ 2, & \text{if } t_i - s_i = 2 \text{ or } 0 \end{cases}$$

To find the string x we use the result of Lemma 3, i.e., $x_i \oplus x_{i+1} = t_i \bullet s_i$. This equation must be satisfied for all i, which leads to the relation $x_i = (\sum (t_i \bullet s_i)) \oplus x_1$. For $\sum (t_i \bullet s_i) = 0$, we will always be able to find a string x satisfying this relation. Hence, we start with an arbitrary value of $x_1 \in \{0, 1\}$. Then successively applying the equation $x_i \oplus x_{i+1} = t_i \bullet s_i$, for $i = 1, 2, \ldots, n-1$, we will get $x_2, x_3, \ldots, x_n$. ■

Let $l(P)$ be the length of a path P of the above form. Now we give the following theorem.

THEOREM 1. *For any pair of nodes (s, t) such that $\sum (t_i \bullet s_i) = 0$, there exists a path P between s and t of the above form such that $l(P) \leq \lfloor (3n)/2 \rfloor$.*

Proof. $l(P) = n + w(x)$, where $w(x)$ is the number of 1's in the binary string x. Now, we can get two strings for x satisfying the equations $x_i \oplus x_{i+1} = t_i \bullet s_i$, one with $x_1 = 0$, and the other with $x_1 = 1$. These two strings are complement to each other and hence for one of them the number of 1's is less than or equal to $\lfloor n/2 \rfloor$. Hence, the proof. ■

Now we consider a path P' of the form $g_1 h_1 g_2 h_2 \cdots h_{n-1} g_n$, where h_i's and g_i's are as defined earlier. Let the path P' from s lead to a node $u = u_1 u_2 \cdots u_n$. Corresponding to the path P' we define two strings $c = c_1 c_2 \cdots c_{n-1}$ and $x = x_1 x_2 \cdots x_n$ as before. Now we state the following lemmas (for proofs see [3]).

LEMMA 6.

$$\text{For } 2 \leq i \leq n,\ u_i = \begin{cases} (s_{i-1} + c_{i-1}), & \text{if } x_i \oplus x_{i+1} = 0 \\ (s_{i-1} + c_{i-1}) + 2, & \text{if } x_i \oplus x_{i+1} = 1. \end{cases}$$

$$\text{Also,} \qquad u_1 = \begin{cases} s_n, & \text{if } x_1 \oplus x_2 = 0 \\ s_n + 2, & \text{if } x_1 \oplus x_2 = 1 \end{cases}$$

LEMMA 7. *For $2 \leq i \leq n$, $x_i \oplus x_{i+1} = u_i \bullet s_{i-1}$, where $x_{n+1} = x_1$.*

LEMMA 8. *For any pair of nodes (s, t) there exists a path of the form P' between s and t', where $t' = t'_1 t_2 t_3 \cdots t_n$, t'_1 being equal to s_n or $s_n + 2$.*

Let $l(s, t')$ denote the length of the path P' between s and t'.

THEOREM 2. $l(s, t') \leq n - 1 + \lfloor n/2 \rfloor$.

Proof. The proof is similar to that of Theorem 1. ■

THEOREM 3. *For any pair of nodes (s, t), there exists a path of length at most $\lfloor (3n)/2 \rfloor + 1$ between s and t.*

Proof. By Theorem 2, $l(s, t') \leq \lfloor (3n)/2 \rfloor - 1$. If $t'_1 = t_1$, then we are done. Otherwise, we consider the following cases.

Case i. $t'_1 = t_1 \pm 1$. If $t'_1 = t_1 + 1$, then we apply $f_2 \varphi_1$ on t' to reach t. If $t'_1 = t_1 - 1$, then we apply $f_1 \varphi_2$ on t' to reach t.

Case ii. $t'_1 = t_1 + 2$. Note that the value of t'_1 is determined by $S = \sum_{i=2}^{n} (t_i \bullet s_{i-1})$. If we could complement S by some means, then the value of t'_1 could be changed from $t_1 + 2$ to t_1. Now consider a node $t_1 t_2 \cdots t_{n-1} t^*_n$ such that $t^*_n \bullet s_{n-1} = (t_n \bullet s_{n-1}) \oplus 1$. Such a t^*_n can always be found out by setting

$$t^*_n = \begin{cases} t_n - 1, & \text{for } (t_n - s_{n-1}) = 1 \text{ or } 3 \\ t_n + 1, & \text{for } (t_n - s_{n-1}) = 0 \text{ or } 2. \end{cases}$$

Thus, starting from s, we can always reach the node $t_1 t_2 \cdots t_{n-1} t^*_n$ by a path of the form $g_1 h_1 g_2 \cdots g_{n-1} h_{n-1} g_n$ having length at most $\lfloor (3n)/2 \rfloor - 1$. Applying now $f_1 \varphi_2$ (if $t^*_n = t_n + 1$) or $f_2 \varphi_1$ (if $t^*_n = t_n - 1$), we can reach $t = t_1 t_2 \cdots t_n$. Hence, the proof. ■

3.1. Comparison with the Moore Bound

According to the Moore bound [11], for a graph of maximum degree d and diameter D, the total number of nodes $N \leq (d(d-1)^D - 2)/(d-2)$, i.e., $D \geq \log_{d-1} N$ (approximately). For $d = 5$, we get $D \geq \log_4 ((3N + 2)/5) \approx 0.5 \log N = D_{\min}$ (say). In our graph the diameter is close to $0.75 \log N$ which is roughly equal to 1.5 times $D_{\min}$. This may be compared with the diameter of other constant degree graphs, e.g., Moebius graph, Cube-Connected Cycle (CCC), De Bruijn grade, as in Table I where N is the number of nodes in each graph.

4. ROUTING ALGORITHM

4.1. Point-to-Point Communication

We can always find a path between a source node s and a destination node t, with a path length less than or equal to the diameter, by employing the method described in the earlier section. The routing algorithm in that case will be simple and its complexity will be no more than that in a Moebius graph [9]. But there may exist a path of much shorter length between s and t than that obtained by this method. This is illustrated in the following example.

TABLE I
Comparison with Three Other Common Graphs

Topology	Degree	Diameter (D) (approx.)	Moore bound ($D_{\min}$) (approx.)	$D/D_{\min}$ (approx.)
Moebius	3	$\frac{3}{4} \log_2 N$	$\log_2 N$	1.5
CCC	3	$\frac{5}{2} \log_2 N$	$\log_2 N$	2.5
de Bruijn	4	$\log_2 N$	$\log_3 N$	1.6
Proposed graph	5	$\frac{3}{4} \log_2 N$	$\log_4 N$	1.5

EXAMPLE 1. Let $s = 01213$ and $t = 01012$. The method given in section 3 gives a path of length 7 as $s \to 12132 \to 21323 \to 21301 \to 13010 \to 30103 \to 01030 \to 01012$.

We may note that there exists a path of length only 2 between s and t given by $s \to 10121 \to 01012 = t$.

In the above example, a path of shorter length was possible because the first 3 digits of s match with the last 3 digits of t. Whenever such a matching exists we can exploit this to get a path of length shorter than that obtained by the method in section 3. Guided by this observation, we discuss below the basic ideas of a near-optimal routing technique.

Let j be the largest integer, $j \geq 1$, such that one of the following conditions hold

(1) $s_{n-i+1} = t_{j-i+1}$, for all $i \in \{1, 2, ..., j\}$
(2) $s_{j-i+1} = t_{n-i+1}$, for all $i \in \{1, 2, ..., j\}$.

Case 1. $s_{n-i+1} = t_{j-i+1}$, for all $i \in \{1, 2, ..., j\}$. Since j bits are already in position, we need only $n - j$ shifts. We consider a path P of the form $h_1 g_1 h_2 g_2 \cdots h_{n-j} g_{n-j}$, where $h_i \in \{f_1, f_2\}$ and $g_i = \mathrm{Id}$ or g. The path P is uniquely characterized by the two strings $c = c_1 c_2 \cdots c_{n-j}$ and $x = x_1 x_2 \cdots x_{n-j}$ as before. For $1 \leq i \leq n - j$, we find c_i by the equation

$$c_i = \begin{cases} 1, & \text{if } t_{i+j} - s_i = 1 \text{ or } 3 \\ 2, & \text{if } t_{i+j} - s_i = 2 \text{ or } 0. \end{cases}$$

Now to find the string x, we set $x_1 = 0$ and then by successively applying the equation $t_{i+j} \bullet s_i = x_i \oplus x_{i+1}$ for $1 \leq i \leq n - j - 1$, we compute $x_2, x_3, ..., x_{n-j}$. Then the path P will lead to the destination node t, if $t_n \bullet s_{n-j} = x_{n-j}$. Otherwise, we will reach a node $t' = t_1 t_2 \cdots t'_n$, where $t'_n = t_n + 2$. Hence, we consider the following subcases.

Subcase 1a. $t_n \bullet s_{n-j} = x_{n-j} \oplus 1$. We will be able to reach the node t by suitably modifying P as done in Case ii of the proof of Theorem 3. Hence the path length l_1 is equal to $n - j + w(x) + 2$.

There is also an alternative path from s to t. Let m, $(m > 1)$ be the first position from the left at which a 1 occurred in the above string, x, i.e., $x_1 = x_2 = \cdots = x_{m-1} = 0$ and $x_m = 1$. This implies that, if $h_{m-1} = f_1$, then $t_{j+m-1} = (s_{m-1} + 3)$. We now modify the string x as follows: we replace h_{m-1} by $f_2 \varphi_1 f_2$, and set x_m to 0 so that t_{j+m-1} is again equal to $(s_{m-1} + 3)$. Similarly if $h_{m-1} = f_2$, then $t_{j+m-1} = s_{m-1}$. We would replace f_{m-1} by $f_1 \varphi_2 f_1$, and set x_m to 0, so that $t_{j+m-1} = a_{m-1}$ again.

Thus, x_m is now 0, whereas earlier it was 1. Keeping h_m, $h_{m+1}, ..., h_{n-j}$ unaltered, we can now get a different string for x as $x_1 x_2 \cdots x_{m-1} \overline{x_m}\ \overline{x_{m+1}} \cdots \overline{x_{n-j}}$. Moreover, we can directly reach the vertex t by this process and do not need the extra two steps at the end as it was necessary corresponding to the earlier string x.

Number of 1's in this new string for x is equal to $(n - j - m) - w(x) + 1$. Hence, the new path length l_2 is equal to $n - j + 2 + n - j - m - w(x) + 1$. The minimum of l_1 and l_2 is less than or equal to $(l_1 + l_2)/2 = (3(n - j + 1) - m)/2 + 1 = (3n)/2 + 1 - (3j + m - 3)/2$ with $j \geq 1$ and $m > 1$. Hence, the path length obtained is always within the bound given for the diameter and *decreases with increase in the value of j*. Also, $l_2 < l_1$ if, $n - j - m - w(x) + 1 < w(x)$; i.e., $w(x) > (n - j - m + 1)/2$.

Subcase 1b. $t_n \bullet s_{n-j} = x_{n-j}$. Here, the path length $l_1 = n - j + w(x)$. An alternative path can also be obtained as in Subcase 1a with 2 more extra steps at the end, resulting in a new path length $l_2 = n - j + 2 + (n - j - m) - w(x) + 3$. The minimum of l_1 and l_2 is again less than or equal to $(l_1 + l_2)/2 = (3(n - j + 1) - m)/2 + 1$. Also, $l_2 < l_1$ if, $(n - j - m) - w(x) + 5 < w(x)$; i.e., $w(x) > (n - j - m + 5)/2$.

Case 2. $s_{j-i+1} = t_{n-i+1}$ for all $i \in \{1, 2, ..., j\}$. Routing is similar to that in Case 1 if we start from t to reach s.

EXAMPLE 2. Let $s = 00000$ and $t = 01011$. By following the method given in Section 3, we find a path of length 8 as $s \to 00001 \to 00012 \to 00121 \to 00103 \to 01032 \to 01010 \to 30101 \to 01011 = t$.

Here $j = 1$, as $s_5 = t_1$ and Case 1 is applicable. By following the routing technique given above, we find a path of length 6 as $s \to 00001 \to 00011 \to 30001 \to 00010 \to 00101 \to 01011 = t$.

Remark. The path of length 2 between s and t of Example 1 also follows from the near-optimal routing technique described above.

Although the path obtained by our routing algorithm may not be the shortest, its length is definitely less than or equal to the bound on diameter of the graph. Also, our routing algorithm requires $O(\log N)$ time for a graph with N nodes. Further, we have computed the shortest paths between every pair of nodes and found that the number of extra links needed by following our routing algorithm, is very small on the average. The results of these computations are given in Table II.

4.2. Single Node Broadcast

To explain the broadcast algorithm, we use the following notations: $a_1 a_2 \cdots a_{n-1}*$ will represent 4 nodes, where $(*)$ can take all possible 4 values 0, 1, 2, and 3. Similarly, by

TABLE II
Average Difference from the Shortest Path Lengths

Number of nodes	Average number of extra links
16	0.4417
64	0.7961
256	1.0300
1024	1.2665
4096	1.5138

using more than one $*$ in different positions, a larger set of nodes can be represented. If x is the number of $*$s in a symbolic notation, then the number of nodes represented will be equal to 4^x. Also, m successive occurrences of the symbol $*$ will be represented by $*^m$.

To broadcast a message from a source node $a_1a_2 \cdots a_n$, we would traverse the nodes in the following sequence: $a_1a_2 \cdots a_n \rightarrow a_2a_3 \cdots a_n* \rightarrow a_3a_4 \cdots a_n** \rightarrow \cdots \rightarrow *^n$

It is to be noted that a single arrow ($\rightarrow$) in the above sequence may involve more than one communication step.

To transfer a message from $a_1a_2 \cdots a_n$ to $a_2a_3 \cdots a_n*$, we note that,

$$f_1(a_1a_2 \cdots a_n) = a_2a_3 \cdots a_n(a_1 + 1)$$
$$f_2(a_1a_2 \cdots a_n) = a_2a_3 \cdots a_n(a_1 + 2)$$
$$f_1\varphi_2 f_1(a_1a_2 \cdots a_n) = a_2a_3 \cdots a_na_1$$
$$f_2\varphi_1 f_2(a_1a_2 \cdots a_n) = a_2a_3 \cdots a_n(a_1 + 3)$$

Hence, it is clear that 3 communication steps will suffice to perform the operations involved in each of the above broadcast steps indicated by an arrow $\rightarrow$. The total number of steps required is equal to $3n$.

For further details, the reader may be referred to [3].

Remark 1. We could have as well broadcast in a reverse way following the sequence as

$$a_1a_2 \cdots a_n \rightarrow *a_1a_2a_3 \cdots a_{n-1} \rightarrow **a_1a_2 \cdots a_{n-2} \rightarrow \cdots \rightarrow *^n$$

For this we just need to replace f_i by φ_i and vice versa in the above procedure.

5. IMPLEMENTATION OF ALGORITHMS

In all discussions that follow, we would represent a processing node intechangeably by its label $a_1a_2 \cdots a_n$ or by a unique integer i in the range 0 to $4^n - 1$, where $i = \sum a_k 4^{n-k}$. Actually, the integer i corresponds to a linear numbering of the nodes with proper weights of the digits a_i's.

5.1. Matrix Transpose

To transpose a $4^m \times 4^m$ matrix A we consider a network of 4^{2m} nodes. Initially the element $A(i, j)$ of the matrix is stored in the node $4^m \times i + j$. After transposing, $A(i, j)$ should be in the node $4^m \times j + i$. In other words, we need to move the element in the node $a_1a_2 \cdots a_ma_{m+1}a_{m+2} \cdots a_{2m}$ to the node $a_{m+1}a_{m+2} \cdots a_{2m}a_1a_2 \cdots a_m$. We consider the following two cases:

Case 1. m is even.

We note that $f_2f_2g(a_1a_2 \cdots a_{2m}) = a_3a_4 \cdots a_{2m}a_1a_2$. Thus, moving the data from a node $a_1a_2 \cdots a_{2m}$ to the node $a_2a_3 \cdots a_{2m}a_1a_2$ will involve 3 communication steps. Applying this sequence of 3 operations successively $m/2$ times, the whole matrix can be transposed. The total number of steps required will be $3(m/2)$.

Case 2. m is odd. Let $m = 2x + 1$.

By applying the sequence of 3 operations (as in Case 1) successively for $\lfloor m/2 \rfloor$ times, the data at the node $a_1a_2 \cdots a_{2m}$ can be moved to the node $a_ma_{m+1} \cdots a_{2m}a_1a_2 \cdots a_{m-1}$.

As $f_1\varphi_2 f_1(a_ma_{m+1} \cdots a_{2m}a_1a_2 \cdots a_{m-1}) = a_{m+1}a_{m+2} \cdots a_{2m}a_1a_2 \cdots a_m$, 3 more communication steps will suffice to move the data to $a_{m+1}a_{m+2} \cdots a_{2m}a_1a_2 \cdots a_m$. The total number of communication steps required in this case is equal to $3\lfloor m/2 \rfloor + 3 = 3\lceil m/2 \rceil$.

For further details, the reader may be referred to [3].

5.2. Maximum/Minimum/Sum/Average

We first consider only the summing up of 4^n elements, where each node contains exactly one element. The maximum/minimum/average can be computed in the same way. We plan to store the final sum in the node $4^n - 1$, i.e. $33 \cdots 3$. We first show below that we can sum 4 elements initially stored in the nodes $*a_2a_3 \cdots a_n$ and put the result in $a_2a_3 \cdots a_n3$, in 3 steps.

Note that,

$$\left.\begin{aligned} f_2(0a_2a_3 \cdots a_n) &= a_2a_3 \cdots a_n2 \\ f_1(1a_2a_3 \cdots a_n) &= a_2a_3 \cdots a_n2 \end{aligned}\right\}. \qquad \text{(A)}$$

Hence, after one communication step, the data stored at the nodes $0a_2a_3 \cdots a_n$ and $1a_2a_3 \cdots a_n$ will move to the node $a_2a_3 \cdots a_n2$. Now we can compute their sum at $a_2a_3 \cdots a_n2$. Also note that,

$$\left.\begin{aligned} f_2(2a_2a_3 \cdots a_n) &= a_2a_3 \cdots a_n0 \\ f_1(3a_2a_3 \cdots a_n) &= a_2a_3 \cdots a_n0 \end{aligned}\right\}. \qquad \text{(B)}$$

Hence, the other two data at $2a_2a_3 \cdots a_n$ and $3a_2a_3 \cdots a_n$ can be summed at $a_2a_3 \cdots a_n0$.

These two sets of operations (A) and (B) can be executed in parallel.

Also,

$$\varphi_1(a_2a_3 \cdots a_n2) = 1a_2a_3 \cdots a_n$$
$$\varphi_2(a_2a_3 \cdots a_n0) = 2a_2a_3 \cdots a_n.$$

Hence, by another communication step, the partial sums can be moved to the nodes $1a_2a_3 \cdots a_n$ and $2a_2a_3 \cdots a_n$, respectively.

Now, applying the steps as in (A), we can obtain the sum in $a_2a_3 \cdots a_n3$.

Summing of 4^n elements involves a sequence of n such operations, which can be represented as

$$*^n \rightarrow *^{n-1}3 \rightarrow \cdots \rightarrow *33 \cdots 3 \rightarrow 33 \cdots 3.$$

We can also proceed in the reverse way as

$$*^n \rightarrow 3*^{n-1} \rightarrow \cdots \rightarrow 33 \cdots 3* \rightarrow 33 \cdots 3.$$

Hence, the sum of 4^n elements can be computed in $3n$ steps of which $2n$ steps involve both communication and computation and the other n steps involve communication only. Note that, the same can be done on a 2-D mesh/torus with degree 4 processors in approximately 2^n steps. On the other hand, although this can be done in $2n$ steps on a $2n$-dimensional hypercube, the degree of each processor in the hypercube would be $2n$ in contrast to just 5 in our proposed topology.

5.3. Matrix Multiplication

Multiplication of matrices on the proposed graph can be done in almost the same way as in a hypercube [A89]. To multiply two matrices A and B of size $4^m \times 4^m$, we take a graph consisting of 4^{3m} nodes. We would represent the processor at the node $a_1a_2 \cdots a_m b_1b_2 \cdots b_m c_1c_2 \cdots c_m$ by $P(x, y, z)$, where $x = \sum a_i 4^{m-i}$, $y = \sum b_i 4^{m-i}$, and $z = \sum c_i 4^{m-i}$. The notation $P(*, y, z)$ is used to denote the set of 4^m processors $P(d, y, z)$, where d assumes all values from 0 to $4^m - 1$. The outline of the multiplication algorithm is given below.

Initially, the elements of the matrix $A[j, k]$ and $B[j, k]$ are stored at the processor $P(0, j, k)$. The elements $C[j, k]$ of the final product matrix $C = AB$ will be stored in the processor $P(r, k, j)$, where $r = 4^m - 1$.

The first step involves a broadcast of $A[j, k]$ and $B[j, k]$ from $P(0, j, k)$ to $P(j, k, *)$ for all j, k, which takes $3m$ steps (each step involves sending both $A[j, k]$ and $B[j, k]$). Now, $P(j, i, i)$ contains $A[j, i]$ and $P(i, k, i)$ contains $B[i, k]$.

Next, we broadcast (in the reverse way) $A[j, i]$ from $P(j, i, i)$ to $P(*, j, i)$ and $B[i, k]$ from $P(i, k, i)$ to $P(*, i, k)$. Hence the processor $P(k, j, i)$ will contain $A[j, i]$ for all k and the processor $P(j, i, k)$ will contain $B[i, k]$ for all j. These two broadcast steps can also be done in $3m$ steps.

Next, the data $B[i, k]$ is moved from $P(j, i, k)$ to $P(k, j, i)$. This involves a shuffle of m bits and by following the method as for matrix transposition, we can do this in time $3\lceil m/2 \rceil$.

Now the processor $P(k, j, i)$ contains $A[j, i]$ and $B[i, k]$. Hence, the product of $A[j, i]$ and $B[i, k]$ can be computed in parallel for all values of i, j, and k. To compute the sum of these products over all i, we require another $3m$ steps and the sums will be available at $P(r, k, j)$; i.e., $P(r, k, j)$ will contain $C[j, k] = \sum A[j, i] \times B[i, k]$. Hence the multiplication can be done in $O(m) = O(\log N)$ steps, where $N \times N$ is the size of each of the matrices A and B. This may be compared with $O(N)$ time on a 2-D mesh/torus and $O(\log N)$ time on a hypercube of degree $3 \log_2 N$.

5.4. ASCEND/DESCEND Algorithms

A class of parallel algorithms, called ASCEND and DESCEND types of algorithm [7] have many areas of application such as FFT and bitonic sorting. Here we will describe the implementation of DESCEND type of algorithms only. We consider N data elements $d[0], d[1], \cdots, d[N-1]$ stored, respectively, at storage locations $x[0], x[1], \cdots, x[N-1]$. We denote a basic operation in the DESCEND type of algorithms by **Oper**$(m, i, x[m], x[m + 2^i])$, where the operation involves the two data elements at locations $x[m]$ and $x[m + 2^i]$ whose binary representations differ in the ith bit. **Oper**$(m, i, x[m], x[m + 2^i])$ computes two values R_0 and R_1, which are stored at $x[m]$ and $x[m + 2^i]$, respectively. Let $N = 4^n = 2^2n$. Then the DESCEND algorithm runs as follows:

for $i = q - 1$ to 0 **do**
  for all m, $0 \le m < N$ **do** in parallel
    if $\text{bit}_i(m) = 0$ **then oper**$(m, i, x[m], x[m + 2^i])$;

Initially, we store the data $d[k]$ at the node $a_1a_2 \cdots a_n$, where a_i's are related to the binary representation $b_1b_2 \cdots b_{2n}$ of k as

$$a_i = 2b_{2i} + b_{2i-1}, 1 \le i \le n.$$

We now give a brief overview of implementing the DESCEND algorithm by our method. We denote by $w[a_1a_2 \cdots a_n]$ the data stored at the node $a_1a_2 \cdots a_n$. Note that,

$$f_2(0*^{n-1}) = *^{n-1}2$$
$$f_1(1*^{n-1}) = *^{n-1}2.$$

Hence, after one communication step the data stored at $0*^{n-1}$ and $1*^{n-1}$ can be moved to $*^{n-1}2$. Then the operation **Oper** can be performed on them. Another communication step will bring the computed results R_0 and R_1 to $0*^{n-1}$ and $1*^{n-1}$, respectively. Similarly, the data stored at $2*^{n-1}$ and $3*^{n-1}$ are operated upon and the results R_0 and R_1 are moved to $3*^{n-1}$ and $2*^{n-1}$, respectively. These two steps complete the operation **Oper** corresponding to the bit b_1. The next step of the DESCEND algorithm corresponding to the bit b_2 will involve data pairs in the nodes $(0*^{n-1}, 3*^{n-1})$ and $(1*^{n-1}, 2*^{n-1})$, respectively.

We note that,

$$f_2(1*^{n-1}) = *^{n-1}3$$
$$f_1(2*^{n-1}) = *^{n-1}3.$$

Thus, the data in $1*^{n-1}$ and $2*^{n-1}$ can be operated upon and stored at $*^{n-1}3$. Now one of the computed results, i.e., R_0 is moved to $*^{n-1}2$ in 2 communication steps as,

$$\varphi_2 f_1(*^{n-1}3) = *^{n-1}2.$$

Simultaneously with these steps the data stored at $3*^{n-1}$ and $0*^{n-1}$ are brought to $*^{n-1}1$ to be operated upon, and the corresponding result R_0 is brought to $*^{n-1}1$.

Hence, after 5 steps, the operation of the DESCEND algorithm corresponding to the most significant two bits

b_1 and b_2 of the binary representation of m can be completed. The above process is repeated n times for completion of the DESCEND algorithm. The detailed steps can be found in [3]. The total time required is equal to $5n$ for 4^n data elements. Note that, on a CCC, the ASCEND/DESCEND algorithm requires roughly $5n$ steps, only for $n2^n$ data elements, in contrast to 4^n data elements on the proposed topology.

6. EXTENSION TO HIGHER DEGREE

We now propose a family of graphs G of maximum degree $2j + 1$, $j \geq 2$ and having $(2j)^n$ nodes, $n \geq 2$. Each node v of the graph is represented by a distinct string of length n, i.e., $v_1v_2 \cdots v_n$, $v_i \in \{0, 1, ..., 2j - 1\}$. For $v \in V$, we define functions $f_k(v)$, $\varphi_k(v)$, $(k \in \{1, 2, ..., j\})$ and $g(v)$ as

$$f_k(v_1v_2 \cdots v_n) = u_1u_2 \cdots u_n,$$
$$\text{where } u_i = \begin{cases} v_{i+1} & \text{for } 1 \leq i \leq n-1 \\ (v_i + k) \bmod 2j, & \text{for } i = n \end{cases}$$
$$\varphi_k(v_1v_2 \cdots v_n) = u_1u_2 \cdots u_n,$$
$$\text{where } u_i = \begin{cases} v_{i-1} & \text{for } 2 \leq i \leq n \\ (v_n - k) \bmod 2j, & \text{for } i = 1 \end{cases}$$
$$g(v_1v_2 \cdots v_n) = u_1u_2 \cdots u_n,$$
$$\text{where } u_i = \begin{cases} v_i & \text{for } 1 \leq i \leq n-2 \\ (v_i + j) \bmod 2j, & \text{for } i = n-1, n. \end{cases}$$

The edge set E of the graph is defined as follows: $(u, v) \in E \Leftrightarrow f_k(u) = v$ or $\phi_k(u) = v$ or $g(u) = v$.

DEFINITION 2. For two digits $a, b \in \{0, 1, ..., 2j - 1\}$, we define an operation $\bullet$ as

$$b \bullet a = \begin{cases} 0, & \text{if } (b - a) \bmod 2j \in \{1, 2, ..., j\} \\ 1, & \text{if } (b - a) \bmod 2j \in \{0, j+1, j+2, ..., 2j-1\}. \end{cases}$$

We consider a path P, starting from the node s and of the form $h_1g_1h_2g_2 \cdots h_ig_i \cdots h_ng_n$, where $h_i = f_k$, $k \in \{1, 2, ..., j\}$, and $g_i = \text{Id}$ or g. Let the path P lead to a node $u = u_1u_2 \cdots u_n$. Corresponding to the path P we define two strings $c = c_1c_2 \cdots c_n$, $c_i \in \{1, 2, ..., j\}$ and $x = x_1x_2 \cdots x_n$, $x_i \in \{0, 1\}$ as

(i) if $h_i = f_k$, $k \in \{1, 2, ..., j\}$, then $c_i = k$
(ii) if $g_i = \text{Id}$, then $x_i = 0$ and if $g_i = g$ then $x_i = 1$.

THEOREM 4. *For any pair of nodes (s, t) such that $\sum (t_i \bullet s_i) = 0$, there exists a path P between s and t of the above form with path length $\leq \lfloor (3n)/2 \rfloor$.*

Proof. Refer to [3]. ■

LEMMA 9. *For any pair of nodes (s, t) there exists a path P' of the form $P' = g_1h_1g_2h_2 \cdots g_ih_i \cdots h_{n-1}g_n$, between s and t', where $t' = t_1't_2t_3 \cdots t_n$, t_1' being equal to s_n or $s_n + j$ and the corresponding path length is upper bounded by $n - 1 + \lfloor n/2 \rfloor$.*

Proof. Refer to [3]. ■

THEOREM 5. *For any pair of nodes (s, t) there exists a path length at most $\lfloor (3n)/2 \rfloor + 1$ between s and t.*

Proof. By Lemma 9, length of the path beween s and t' is $\leq \lfloor (3n)/2 \rfloor - 1$. If $t_1' = t_1$, then we are done. Otherwise, we consider the following cases.

Case i. $t_1' = t_1 \pm k$, $k \in \{1, 2, ..., j - 1\}$. If $t_1' = t_1 + k$, then we apply $f_{k+1}\varphi_1$ on t' to reach t. If $t_1' = t_1 - k$, then we apply $f_1\varphi_{k+1}$ on t' to reach t.

Case ii. $t_1' = t_1 + j$. Note that the value of t_1' is determined by $S = \sum_{i=2}^{n} (t_i \bullet s_{i-1})$. If we could complement S by some means, then the value of t_1' could be changed from $t_1 + j$ to t_1. Now consider a node $t_1t_2 \cdots t_{n-1}t_n^*$ such that $t_n^* \bullet s_{n-1} = (t_n \bullet s_{n-1}) \oplus 1$. Such a t_n^* can always be found out by setting

$$t_n^* = \begin{cases} t_n - k, & \text{for } (t_n - s_{n-1}) = k \text{ or } j + k, k \in \{1, 2, ..., j-1\} \\ t_n + 1, & \text{for } (t_n - s_{n-1}) = 0 \text{ or } j. \end{cases}$$

Thus, starting from s, we can always reach the node $t_1t_2 \cdots t_{n-1}t_n^*$ by a path of the form $g_1h_1g_2 \cdots g_{n-1}h_{n-1}g_n$ having length at most $\lfloor (3n)/2 \rfloor - 1$. Applying now $f_1\varphi_2$ (if $t_n^* = t_n + 1$) or $f_{k+1}\varphi_1$, $k \in \{1, 2, \cdots j - 1\}$ (if $t_n^* = t_n - k$), we can reach $t = t_1t_2 \cdots t_n$. Hence, the proof. ■

7. CONCLUSIONS

We have proposed a new family of graphs with $(2j)^n$ nodes, $j \geq 2$, $n \geq 2$, with a maximum node degree of $2j + 1$. The diameter of this graph is bounded above by $\lfloor (3n)/2 \rfloor + 1$, which is, however, not tight. For the graphs of maximum degree 5, we have exhaustively checked that except for $n = 5$, the diameter is one less than this bound for all values of n less than 7. For $n = 5$, the diameter is equal to this bound and the diameter may be equal to or less than this bound for $n > 6$. Implementation of various algorithms on this topology has been discussed. Future works may include studying the fault-tolerance properties of the topology and embedding of other popular networks such as trees and meshes.

REFERENCES

1. S. B. Akers and B. Krishnamurti. Group graphs as interconnection networks. In *Proc. Int. Conf. Fault-tolerant Comput.* 1984, pp. 422–427.
2. N. G. de Bruijn. A combinatorial problem. *Koninklijke Nederlande Academie van Watenschappen Proc.*, 1946. Vol. A49, pp. 758–764.
3. R. K. Das. Design, analysis and routing in static interconnection networks. Ph.D. thesis, Indian Statistical Institute, Calcutta, 1995.
4. C. C. Reames and M. T. Liu. A loop network for simultaneous

transmission of variable length messages. *Proc. Second Symp. Computer Architecture*. Jan. 1975, pp. 7–12.

5. J.-C. Bermond, and D. Tzvieli. Minimal-diameter double-loop networks: Dense optimal family. *Networks* **21** (Jan. 1991), 1–9.
6. F. Harary. *Graph Theory*. Addison-Wesley, New York, 1969.
7. F. P. Preparata and J. Vuillemin. The cube-connected cycles: A versatile network for parallel computation. *Comm. ACM* **24** (May 1981), 300–309.
8. W. H. Kautz. Design of optimal interconnection networks for multiprocessors. In *Architecture and Design of Digital Computers*, Nato Advanced Summer Institute, 1969, pp. 249–272.
9. W. E. Leland and M. H. Solomon. Dense trivalent graphs for processor interconnection. *IEEE Trans. Comput. C* **31**(3) (Mar. 1982), 219–222.
10. B. W. Arden and H. Lee. Analysis of chordal ring network. *IEEE Trans. Comput. C* **30** (Apr. 1981), 291–295.
11. E. Bannai and T. Ito. On finite Moore graphs. *J. Fac. Sci. Univ. Tokyo* (1973), 191–208.

RAJIB K. DAS received his B.E. in electronics and telecommunication engineering from Jadavpur University in 1988, his M.Tech. in computer science from the Indian Institute of Technology, Madras in 1990, and his Ph.D. in computer science from the Indian Statistical Institute, Calcutta in 1995. Currently he is a reader in the Department of Computer Science in Tezpur University, Assam, India. His research interests include network topology and parallel and distributed computing.

BHABANI P. SINHA received his B.Sc. in physics, B.Tech. and M.Tech. degrees in radiophysics and electronics, and Ph.D. in computer science from the University of Calcutta in 1970, 1973, 1975, and 1979, respectively. In 1976, he joined the faculty of the Indian Statistical Institute, Calcutta, where he has been a professor since 1987. During 1986–1987, he visited the Department of Computer Science, Southern Illinois University, as an associate professor. With a fellowship from the Alexander von Humboldt Foundation of West Germany, he visited Informatik Kolleg, GMD, Bonn from 1979 to 1981 and also the Department of Computer Science, University of Saarland, Saarbruecken in 1990. He has published more than 80 research papers in the areas of microprogramming, computer arithmetic, system diagnosis, network topology, parallel algorithms, and applied graph theory. His current research interests include parallel algorithms and architectures, network topology, and computational geometry. He is a senior member of the Institution of Electrical and Electronics Engineers and the IEEE Computer Society.

Received October 11, 1995; revised June 24, 1996; accepted June 25, 1996