CARTESIAN PRODUCT NETWORKS

Abdou Youssef
Department of Electrical Engineering and Computer Science
The George Washington University
Washington, DC 20052

Abstract –This paper will study product graphs as interconnection networks. The topological properties of product networks will be presented, and generic, divide-and-conquer algorithms for point-to-point routing, broadcasting and permuting will be designed. Finally, linear arrays, rings, meshes, toruses and trees will be embedded on product networks.

Introduction

In this paper a unified framework is developed in which most existing networks and many new ones can be studied. The class of cartesian product graphs which will be called here product networks, will provide this common framework. Multidimensional meshes, multidimensional toruses, binary and generalized k-ary hypercubes and others belong to this class. Other useful product network that can be extended with fixed node degree will be proposed. The paper will study the topological properties of product networks and develop various generic routing and embedding algorithms for them.

Product Networks

In this section product graphs will be reviewed and their topological properties will be presented.

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two graphs. The product graph of G_1 and G_2, denoted $G_1G_2 = (V_1V_2, E)$, is a graph where the set of nodes is the product set $V_1V_2 = \{x_1x_2 \mid x_1 \in V_1 \text{ and } x_2 \in V_2\}$ and $E = \{\langle x_1x_2, y_1, y_2\rangle \mid (x_1 = y_1 \text{ and } \langle x_2, y_2\rangle \in E_2) \text{ or } (x_2 = y_2 \text{ and } \langle x_1, y_1\rangle \in E_1)\}$. The graphs G_1 and G_2 are called the factors of G_1G_2.

As can be observed, G_1G_2 consists of $|V_2|$ copies of G_1 where every set of the $|V_1|$ corresponding nodes of these copies form a G_2 graph. The copy of G_2 in G_1G_2 that corresponds to a node $x_1 \in V_1$ is denoted x_1G_2. Its set of nodes is $\{x_1x_2 \mid x_2 \in V_2\}$ and its set of edges is $\{\langle x_1x_2, x_1y_2\rangle \mid \langle x_2, y_2\rangle \in E_2\}$. Similarly, The copy of G_1 in G_1G_2 that corresponds to a node $x_2 \in V_2$ is denoted G_1x_2.

Note that the product $G_1G_2G_3...G_n = (V, E)$ of n networks $G_1 = (V_1, E_1), G_2 = (V_2, E_2), ..., G_n = (V_n, E_n)$ can be derived. Clearly, $V = V_1V_2V_3...V_n$ and $E = \{\langle x_1x_2...x_n, y_1y_2...y_n\rangle \mid$ there exists an i such that $\langle x_i, y_i\rangle \in E_i$ and for every $j \neq i$ we have $x_j = y_j\}$. If all G_i's are equal (to some G), $G_1...G_n$ is denoted G^n.

Denote by L_p, R_p and K_p a linear array of p nodes, a ring of p nodes, and a complete graph of p nodes, respectively. It can be seen that a $p_1 \times p_2 \times ... \times p_n$ mesh (resp., torus) is $L_{p_1}L_{p_2}...L_{p_n}$ (resp., $R_{p_1}R_{p_2}...R_{p_n}$). Similarly, an n-cube Q_n is the product network k_2^n. The n-cube of radix $r \geq 2$ is K_r^n.

A network is said to be *indefinitely extendable* if it can be extended without increasing the node degree. Linear arrays, rings, and meshes are indefinitely extendable but hypercubes are not. As $degree(G_1G_2) = degree(G_1) + degree(G_2)$, we conclude that if G_1 is indefinitely extendable then the product G_1G_2 is indefinitely extendable. Note that G_2 does need not be indefinitely extendable. Thus, the **line-hyeprcube** L_rQ_n, the **ring-hypercube** R_rQ_n and the **mesh-hypercube** $L_pL_qQ_n$ are indefinitely extendable to L_sQ_n, R_sQ_n and $L_{p'}L_{q'}Q_n$ respectively, for any $s > r$, $p' \geq p$, $q' \geq q$. Hence, the product operator provides an excellent cube augmentation scheme. It has also other desirable properties as summarized below.

Denote by $d_G(x,y)$ the distance from x to y in G, by D_G the diameter of G, by $\overline{d}_G$ the average distance of G, and by C_G the connectivity of G, that is, 1+the largest number of nodes that can be deleted without disconnecting G.

Theorem 1. Let G_1 and G_2 be two graphs.
1) $d_{G_1G_2}(xx', yy') = d_{G_1}(x,y) + d_{G_2}(x', y')$.
2) $D(G_2G_2) = D(G_1) + D(G_2)$.
3) $\overline{d}_{G_1G_2} = \overline{d}_{G_1} + \overline{d}_{G_2}$.
4) $C_{G_1G_2} \geq C_{G_1} + C_{G_2}$.

Proof. See [5].

In part (4) above, the equality holds for some products but not for others. As a corollary of this theorem, the diameter, average distance and connectivity of meshes, toruses and hypercubes can be rediscovered and these same measures can be concluded for the line-hypercube, ring-hypercube and mesh-hypercube.

Routing

Let G_1 and G_2 be two networks such that each G_i is endowed with a point-to-point routing algorithm ROUTE$_{G_i}(s,d)$ which sends a message from s to d, a broadcasting algorithm BROADCAST$_{G_i}(s, M)$ which broadcasts a message M from s to all other nodes, and a permuting algorithm PERMUTE$_{G_i}(f)$ which routes a message from every node x to node $f(x)$, where f is a permutation. These algorithms will be used to devise corresponding algorithm for G_1G_2 in a divide-and-conquer fashion.

procedure ROUTE$_{G_1G_2}(s_1s_2, d_1d_2)$
begin
  ROUTE$_{G_1}(s_1s_2, d_1s_2)$;
  ROUTE$_{G_2}(d_1s_2, d_1d_2)$;
end

procedure BROADCAST$_{G_1G_2}(s_1s_2, M)$
begin
  1. BROADCAST$_{G_1s_2}(s_1s_2, M)$;
  2. **forall** nodes x in G_1 **do in parallel**
    BROADCAST$_{xG_2}(xs_2, M)$;
end

Note that if each ROUTE$_{G_i}$ and BROADCAST$_{G_i}$ is optimal in time, then ROUTE$_{G_1G_2}$ and BROADCAST$_{G_1G_2}$ are optimal.

Next, the more elaborate permutation routing will be addressed. Our approach is based on Clos routing [3]. We review Clos networks first. A Clos network $C(p,q)$ has pq input terminals, pq output terminals and three columns of switches. The 1st and 3rd columns have p $q \times q$ crossbar switches each. The 2nd column has q $p \times p$ crossbar switches. The switches in each column are labeled 0, 1, ... The input ports and the output ports of every switch x are labeled xy ($y = 0, 1, ...$) so that xy is the y-th port of switch x. The interconnection between the first two columns links output port xy in the 1st column to input port yx in the 2nd column. Similarly, the interconnection between the last two columns links output port yx in the 2nd column to input port xy in the 3rd column. It has been shown in [1] that every permutation of pq elements is realizable by $C(p,q)$ in $O((pq)^2)$ time, that is, for every permutation f, there are switch settings for all the switches so that the source-destination paths $i \rightarrow f(i)$ are established without conflict. Call the algorithm that determines the switch setting $\mathrm{CLOS}_{pq}(f)$.

Let xy be an arbitrary source and $xy \rightarrow f(xy)$ the corresponding source-destination path established by $CLOS_{pq}(f)$. The detailed parts of this path are:

$$xy \rightarrow xy' \rightarrow y'x \rightarrow y'x' \rightarrow x'y' \rightarrow x'y'' = f(xy)$$

for some x', y' and y''. Denote by f_x the permutation that maps y to y' in switch x of the 1st column, by $g_{y'}$ the permutation that maps x to x' in switch y' of the 2nd column, and by $h_{x'}$ the permutation that maps y' to y'' in switch x' of the 3rd column.

Suppose that f is to be routed in G_1G_2, where G_1 has p nodes labeled $0, 1, ..., p-1$, and G_2 has q nodes labeled $0, 1, ..., q-1$. By corresponding every node xy in G_1G_2 to input/output terminal xy of $C(p,q)$, every xG_2 to switch x in the 1st/3rd column of $C(p,q)$, and every G_1y to switch y in the 2nd column of $C(p,q)$, we obtain an algorithm for G_1G_2:

Procedure $\mathrm{PERMUTE}_{G_1G_2}(f)$

1. Let $\mathrm{CLOS}_{pq}(f)$ determine the f_x's, $g_{y'}$'s and $h_{x'}$'s.
2. $\mathrm{PERMUTE}_{xG_2}(f_x)$ for all x in parallel
3. $\mathrm{PERMUTE}_{G_1y'}(g_{y'})$ for all y' in parallel
4. $\mathrm{PERMUTE}_{x'G_2}(h_{x'})$ for all x' in parallel

Proof of correcteness: Let M_{xy} be the message to be send from the arbitrary node xy to node $f(xy)$. Let x', y' and y'' be as before. After routing f_x on xG_2, M_{xy} is at node xy'. After routing $g_{y'}$ on G_1y', M_{xy} is at node $g_{y'}(x)y' = x'y'$. Finally, after routing $h_{x'}$ on $x'G_2$, the message M_{xy} is at node $x'h_{x'}(y') = x'y'' = f(xy)$.

The major drawback of this algorithm is the inefficiency of the CLOS algorithm. However, we have developed a new approach to self-routing on CLOS networks [4]. This approach is based on the observation that if the 1st column in a Clos network is set to some configuration, the resulting network becomes self-routed using destination addresses. Accordingly, the approach seeks, for every given family of permutations, a configuration to which to set the first column so that the resulting delta network realizes all the permutations of the family. Such configuration of the first column is called the compatibility factor. Compatibility factors were found in [4] for several important families of permutations. These include the families of permutations required by FFT, bitonic sorting, tree computations, multidimensional mesh/torus computations, multigrid computations [2] as well as the Omega permutations.

It will be argued next that if the compatibilty factor is known, then $\mathrm{PERMUTE}_{G_1G_2}$ becomes a self-routing algorithm and step 1 is bypassed. The f_x's are clearly the compatibility factor and hence known. $g_{y'}(x) = x'$ =the first part of the destination address $x'y'' = f(xy)$ of the message M_{xy}. Thus, node xy' can alone determine its intermediate destination $x'y'$. $h_{x'}(y') = y''$ =the second part of the destination address of M_{xy}. Thus, node $x'y'$ can alone determine the destination $x'y''$.

A noteworthy special case is the Omega-realizable permutations. Their compatibility factor is the identity permutation. That is, the f_x's for every Omega-realizable permutation f are identity permutations and therefore, do not have to be routed. Thus, step 1 and 2 can be bypassed in this case while steps 3 and 4 are executable in a distributed manner.

Next we evaluate the communication complexity of permuting using PERMUTE and taking the number of conflict-free steps as the complexity measure.

Theorem 2. Let $P(G)$ be the minimum number of steps needed for permuting in G. then,

1) $P(G_1G_2) \leq \mathrm{MIN}(2P(G_1) + P(G_2), 2P(G_2) + P(G_1))$.

2) $P(G_1G_2...G_k) \leq 2\sum_{i=1}^{k} P(G_i) - \mathrm{MAX}_{i=1}^{k}(P(G_i))$

3) For Ω-realizable permutations, $P(G_1G_2...G_k) \leq \sum_{i=1}^{k} P(G_i)$

The proof follows directly from the algorithm and the above discussion. We can then conclude: $P(p \times p \text{ mesh}) \leq 3(p-1)$, $P(K_r^n) \leq 2n - 1$ and $P(L_rQ_n) \leq \mathrm{MIN}(2r + 2n - 4, 4n + r - 5)$.

Embedding in Product Networks

An embedding of a guest graph $G = (V_g, E_g)$ on a host graph (i.e., network) $H = (V_h, E_h)$ is a mapping f from V_g to V_h. The standard goodness measure of a mapping f is the dilation cost: $\mathit{dilation}(f) = \mathrm{MAX}\{d_{G_h}(f(x), f(y)) \mid (x,y) \in E_g\}$. We will limit ourselves to one-to-one embeddings, that is, to cases where f is one-to-one. In this section, embeddings for lines, rings, meshes, toruses and trees will be constructed on product networks using the embeddings of these structures on the factor networks. We first start with a general theorem.

Theorem 3. If for $i = 1, ..., k$ the graph G_i can be embedded on the graph H_i with a mapping f_i of dilation d_i, then $G_1G_2...G_k$ can be embedded on $H_1H_2...H_k$ with dilation $\mathrm{MAX}(d_1, ..., d_k)$ with the mapping $f(x_1...x_k) = f_1(x_1)...f_k(x_k)$.

Therefore, if a linear array L_{p_i} (resp., ring R_{p_i}) can be embedded on H_i with dilation cost d_i, for every i, then the k-dimensional $p_1 \times p_2 \times ... \times p_k$ mesh (resp., torus), can be embedded on $H_1H_2...H_k$ with dilation cost $\mathrm{MAX}(d_1, ..., d_k)$. In particular, by using the standard embedding of linear arrays and rings in meshes and toruses, one can achieve an embedding of a linear array or a ring in the product graph.

Next we embed a tree in G_1G_2. Assume that T_i is a tree of hight h_i that can be embedded in G_i with dilation d_i $(i = 1, 2)$. Let x_2 be a the root node of T_2 in G_2. Embed the tree T_1x_2 in G_1x_2, then embed the tree x_1T_2 of root x_1x_2 in x_1G_2 for all nodes x_1 of G_1. The resulting embedded structure $T_1x_2 \cup \cup_{x_1 \in G_1}\{x_1T_2\}$ is clearly a tree of hight $h_1 + h_2$ and the dilation cost of the embedding is $\mathrm{MAX}(d_1, d_2)$.

Conclusions

A theory of product interconnection networks has been developed in this paper. Product networks were shown to include many of the existing networks as well as other useful, indefinitely extendable networks. It was shown that as the number of nodes grows multiplicatively, the degree, diameter, average distamce and connectivity grow additively. Finally, product networks were constructively shown to yield to a divide-and-conquer approach of routing and embedding.

References

[1] V. E. Benes, *Mathematical theory on connecting networks and telephone traffic*, Academic Press, New York, 1965.

[2] T. F. Chan and Y. Saad, "Multigrid algorithms on the Hypercube multiprocessor," *IEEE Trans. Comput.*, vol. C-35, pp. 969–977, Nov. 1986.

[3] C. Clos, "A Study of Non-Blocking Networks," *Bell System Technical Journal 32*, pp. 494–603.

[4] A. Youssef and B. Arden, "A New Approach to Fast Control of $r^2 \times r^2$ 3-Stage Benes Networks of $r \times r$ Crossbar Switches," *Proc. of the 17th Annual Int'l Symp. on Computer Architecture*, pp. 50–59, May 1990.

[5] A. Youssef, "Product Networks: A Unified Theory of Fixed Interconnection Networks," Technical Report GWU-IIST-90-38, The George Washington University, Dec. 1990.