A Unified Approach to Algorithm Development for Product Networks[*]

Antonio Fernández[†] Kemal Efe[‡] Adrienne L. Broadwater[‡] M. Araceli Lorenzo[§] Daniel Calzada[§]

ABSTRACT

There is a major problem with algorithm portability when the user switches from one parallel architecture to another. Since algorithms are usually architecture-dependent, the algorithm running on the old architecture may not run on the new one. Standard techniques, like parallelizing compilers or emulation, have efficacies far below those of algorithms specifically developed for the individual architecture.

This paper proposes a two-level approach to programming parallel computers that is applicable as long as the underlying interconnection architecture can be modeled as a product network (e.g. grid, torus, hypercube, etc.). Our approach assumes that there are some low-level routines optimized for the "factor" networks comprising the product network. The set of low-level routines can be implemented as library routines. The high-level programming is then achieved, oblivious to the topology of the factor networks, by decomposing computations in a manner that only uses the set of low-level routines. We show how to decompose several problems, including matrix multiplication, pointer jumping, FFT, computing the transitive closure of a graph, as well as several other graph-theory problems. The analyses of running time complexities show that, for several product networks, these algorithms are either optimal or they match the complexity of the fastest-known algorithms specifically developed for these networks. As the class of product networks contains existing architectures such as hypercubes and grids, the results of this paper have significant practical value.

Keywords: parallel architectures, parallel programming, FFT, graph algorithms, matrix multiplication, routing, pointer jumping.

INTRODUCTION

Users of massively parallel computers face a major problem when they change their hardware platforms: an algorithm developed for the older architecture will not run on the new architecture. To remedy this problem, specialized languages equipped with parallelization routines (such as High Performance Fortran) have been developed, but the efficiency of the parallel code generated by these compilers is far below the efficiency of algorithms developed specifically for each individual

[*]This research has been partially supported by a grant from the Louisiana Board of Regents, contract no: LEQSF-(1995-97)-RD-A-33.

[†]Grupo de Comunicaciones, Dpto. de Arquitectura y Tecnología de Computadores, U. Politécnica de Madrid, Spain. anto@eui.upm.es. Partially supported by the Spanish Ministry of Education.

[‡]Center for Advanced Computer Studies, U. of Southwestern Louisiana, Lafayette, LA.

[§]Grupo de Comunicaciones, Dpto. de Arquitectura y Tecnología de Computadores, U. Politécnica de Madrid, Spain

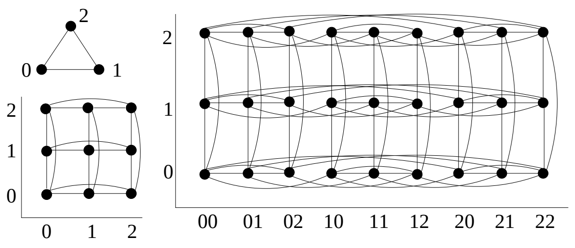

Figure 1: Construction of product networks.

architecture. Other standard techniques, like emulations via embeddings, also incur in significant slowdown.

This paper proposes a more direct approach to programming parallel computers that is applicable as long as the underlying interconnection architecture can be modeled as a product network (e.g. grid, torus, hypercube, etc.). Rather than using a parallelizing compiler to map the sequential code, we advocate using high-level algorithms that are written in a manner oblivious to the underlying network topology. Two basic assumptions behind our approach are that (a) there exist some low-level routines optimized for the "factor" network comprising the product network, and (b) there exist an abstract level specification of work to be done to solve a problem, given that certain low level functions have been implemented as in the first assumption. The lower level routines perform a set of basic functions and they can be implemented as library routines by the manufacturer of the hardware, or by compiler designers. Showing the validity of the second assumption for several important problems is the main contribution of this paper. In particular, we show how to decompose several computations in a manner that only uses a set of generic routines that we assume to be available, and optimized for the "factor" network.

In the literature, generic algorithms running on product networks have been presented for several problems: sorting [5], broadcasting and point-to-point communication [12], permutation routing [1], and fault-tolerant routing. [4, 8]. This paper enhances this list by introducing new algorithms for matrix multiplication, computing the transitive closure of a graph, and Fast Fourier Transform. We compute the running time complexity of these algorithms for several instances of product networks and show that the running time is either optimal, or it matches the complexity of the fastest-known algorithms specifically developed for these systems.

DEFINITIONS AND NOTATION

We first define the cartesian product of two graphs. Given a graph G, we use $V(G)$ to denote its vertex set and $E(G)$ to denote its edge set.

Definition 1 *The product of two graphs G and H is the graph GH, whose vertex set is $V(G) \times V(H)$, and whose edge set is defined as follows: Assume $x, x' \in V(G)$ and $y, y' \in V(H)$, then $((x,y),(x',y'))$ is an edge in GH if and only if either $x = x'$ and $(y,y') \in E(H)$, or $y = y'$ and $(x,x') \in E(G)$.*

Construction of a two-dimensional product network GH according to this definition is illustrated in Figure 1. We first draw H with all its nodes aligned horizontally. Suppose G has n_G nodes, then we make n_G copies of the H graph drawn in such a way that nodes with identical labels fall in the same column. Each column is then connected according to the interconnection pattern of G. The resulting product network can be seen as a two-dimensioal grid in which the row connections have been substituted by the connections of H and the column connections have been substituted by those of G.

We can similarly construct a three dimensional product network GHI. In this case we would first construct the two-dimensional product GH. Then, if I has n_I nodes, we would make n_I copies of GH, and use these as the planes connected in the third dimension. The third dimension connections are formed according to the interconnection pattern of I. Observe that the resulting 3-dimensional network can be seen as a 2-dimensional product network in which one of the factor networks is itself a product network. We can similarly construct product networks with several dimensions.

Given the product network GH, by construction each row is a subgraph isomorphic to H and each column is a subgraph isomorphic to G. Furthermore, each of the H subgraphs in a row can be uniquely identified with a label $x \in V(G)$, since all its nodes have x as the left component in the label. Thus, we shall use xH to denote the row subgraph H of GH identified with label x. For example, $2H$ is the H subgraph in the top row of Figure 1. Similarly, each column can be identified with a label $y \in V(H)$, and Gy denotes the column subgraph G identified with label y. We use xy to refer to a specific node in the two dimensional product graph, where x denotes the column position of the node, and y denotes the row position.

In a three dimensional product network GHI the above notation can be naturally extended to denote its corresponding subgraphs. For instance, GHz would denote a 2-dimensional subgraph of GHI, isomorphic to GH, and identified by a label $z \in V(I)$.

In several places in the paper we need to refer to array variables contained in the nodes of a network G, one array element for each node. To do so, we shall use the notation $a[x]$, where a is a one dimensional array and x is the index variable, $x \in V(G)$. We will use Greek letters such as α, β to indicate specific constants that these index variables take. For example, while $a[x]$ refers to an array, $a[\alpha]$ refers to a specific element in the array. We will similarly extend these notations for two or more dimensions.

ROUTING, BROADCASTING, AND SUMMATION

We first illustrate the above notations by presenting some routing algorithms due to Youssef [12]. We will subsequently introduce a new algorithm for summation of values held in procesors (one for each processor). These will form a collection of "building blocks" that other algorithms will use, either directly or in a slightly modified form.

In a **point-to-point routing** problem a source node α sends a packet s to a target node β in the factor network. Assume that such an algorithm exists for two factor networks G and H. Here we are interested in using such primitive operations to construct a point-to-point routing algorithm for the two-dimensional product GH. More formally:
Given:

```
Operation Route($G$: Network; $\alpha$: Source; $\beta$: Destination; $s$: Data)
    Let $p[x]$ be an array stored in the nodes of $G$, one each.
    Before: $p[\alpha] = s$, where $s$ is the value to be sent.
    After: $p[\beta] = p[\alpha] = s$.
```

The point-to-point routing algorithm for GH is:

```
Algorithm Product-Route($GH$: Network; $\alpha\beta$: Source; $\alpha'\beta'$: Destination; $s$: Data)
    Route($G\beta$, $\alpha\beta$, $\alpha'\beta$, $s$);
    Route($\alpha' H$, $\alpha'\beta$, $\alpha'\beta'$, $s$)
```

Recall that $G\beta$ refers to the G subgraph of GH in column β of the product network. The first procedure call "Route($G\beta$, $\alpha\beta$, $\alpha'\beta$, s)" sends the message s from $\alpha\beta$ to $\alpha'\beta$ in $G\beta$. The last line of the algorithm sends the data s from $\alpha'\beta$ to $\alpha'\beta'$ in the H graph at row α'.

In a **broadcasting** operation a source node α sends a copy of the same packet $p[\alpha]$ to all the nodes in the network.
Given:

```
Operation Broadcast($G$: Network; $\alpha$: Source; $s$: Data)
    Let $p[x]$ be an array of packets
    Before: $p[\alpha] = s$, where $s$ is the value to be broadcast.
    After: $\forall x \in V(G)$, $p[x] = p[\alpha] = s$.
```

The algorithm to broadcast in a product network is given as follows.

```
Algorithm Product-Broadcast($GH$: Network; $\alpha\beta$: Source; $s$: Data)
    Broadcast($G\beta$, $\alpha\beta$, $s$);
    For each $x \in V(G)$ do in parallel: Broadcast($xH$, $x\beta$, $s$)
```

At the end of the first "Broadcast" operation, every node in column β has a copy of the message which was originally in processor $p[\alpha\beta]$. The second "Broadcast" operation broadcasts these values at each row in parallel. If the primitive routing algorithms for G and H take optimal time, then the above algorithm also takes optimal time.

A **summation** algorithm obtains the sum of the values initially stored in the nodes of a network, one for each node, and leaves the result in some specific node α. Formally:
Given:

```
Operation Sum($G$: Network; $\alpha$: Node; $a[x]$: Array)
    Before: $\forall x \in V(G)$, $a[x] = v_x$, where $v_x$ is the value to be added.
    After: $a[\alpha] = \sum v_x$;   $for\ all\ x \in V(G)$.
```

This algorithm overwrites the sum on the old value $a[\alpha]$ in processor α. If overwriting is not desired, a new variable can be introduced to contain the final value. The following algorithm shows how to compute the sum in the product network, leaving the result in the node $\alpha\beta$.

```
Algorithm Product-Sum($GH$: Network; $\alpha\beta$: Node; $a[xy]$: Array)
    For each $x \in V(G)$ do in parallel: Sum($xH$, $x\beta$, $a[xy]$);
    Sum($G\beta$, $\alpha\beta$, $a[x\beta]$)
```

Here the first "For each" statement applies a "Sum" operation in each row. Due to the fact that xH denotes the factor graph H in row x, the resulting row sum resides at processor $x\beta$, for all $x \in G$. The only remaining work is to sum the new values at processors of column β. The last "Sum" operation sums these values, storing the result in processor $\alpha\beta$. If we use $S(G)$ and $S(H)$ to denote the complexity of the summation algorithm on networks G and H respectively, this algorithm has complexity $S(GH) = S(G) + S(H)$.

Several problems have a structure similar to that of the summmation problem (obtaining the product, the maximum, the minimum, etc.). Algorithms for these problems can be obtained with straightforward modifications of the summation algorithm presented here.

ADVANCED ALGORITHMS

Pointer Jumping: Let u be a vector of n values, the pointer-jumping operation obtains a new vector w, such that $w[i] = u[u[i]]$, for $i = 0, ..., n-1$. Below we present an algorithm for a more general version of the pointer-jumping operation. We initially have two vectors u and v, of n values, and we obtain the vector w such that $w[i] = u[v[i]]$, for $i = 0, ..., n-1$. The special case of $u = v$ gives the traditional pointer jumping operation.

Given: Above algorithms for point-to-point routing and broadcasting in factor networks.

In the algorithm below the "$\circ$" represents some "signal" sent from one node to another.

```
Algorithm Pointer-Jumping($GH$: Network; $xy$: Node; $u[y], v[x], w[y], p[xy]$: Array)
    Do in parallel
        For each $x \in V(G)$ do in parallel: Broadcast($xH$, $x0$, $u[x0]$;)
        Each processor $xy$ stores the received value in $p[xy]$
        For each $y \in V(H)$ do in parallel: Route($Gy$, $0y$, $v[x]y$, $\circ$)
    For each $x \in V(G)$, $y \in V(H)$ do in parallel:
        If Received $\circ$ then Route($Gy$, $xy$, $0y$, $p[xy]$);
        Each processor $0y$ in $0H$ stores the received value in $w[y]$
```

Assuming that the $u[y]$ vector is initially held in $G0$ subgraph of GH, the first "For each" operation broadcasts the $u[y]$ vector using all the H subgraphs in the rows, independently. Assuming that $v[x]$ is initially held in the $0H$ subgraph of GH, the second "For each" operation sends the "$\circ$" signals to $v[x]$th processor in each G graph in the columns. During the last "For each" operation, processors receiving this "$\circ$" message send their values to processor 0 in their column. These values are then stored as the $w[y]$ vector at subgraph $0H$. Let $B(H)$ denote the time to broadcast on H, and $R(G)$ denote the time to route on G. Then, this algorithm has complexity $P(GH) = \max(B(H), R(G)) + R(G) + 1$.

Matrix Multiplication: Let A and B be the matrices to be multiplied. We present an algorithm that computes the product matrix C in a three dimensional network GHI. We assume that A is a n_G by n_H matrix, while B is a n_H by n_I matrix. Then, C is a n_G by n_I matrix. If $a_{i,k}$ are the elements of matrix A, and $b_{k,j}$ are the elements of matrix B, then the element $c_{i,j}$ of C is obtained as $c_{i,j} = \sum_{k=1}^{n_H} a_{i,k} b_{k,j}$. Here we assume $V(G) = \{1, ..., n_G\}$, $V(H) = \{1, ..., n_H\}$, and $V(I) = \{1, ..., n_I\}$. Initially, the A and B matrices are stored in $GH0$, and $0HI$, respectively. The resulting C matrix will be obtained in $G0I$.

Given: Above algorithms for broadcasting and summation in factor networks.

```
Algorithm Matrix-Multiply($GHI$: Network; $xyz$: Node; $A[xyz], B[xyz], C[xyz]$: Array)
Do in parallel
    For each $x \in V(G), y \in V(H)$ do in parallel:
        Broadcast($xyI$, $xy0$, $A[xy0]$);
    For each $y \in V(H), z \in V(I)$ do in parallel:
        Broadcast($Gyz$, $0yz$, $B[0yz]$);
For each $x \in V(G), y \in V(H), z \in V(I)$ do in parallel:
    $C[xyz] = A[xyz]B[xyz]$;
```

```
For each $x \in V(G), z \in V(I)$ do in parallel:
    Sum($xHz$, $x0z$, $C[xyz]$)
```

The algorithm works by multicasting the elements of the matrices A and B so that their product can be computed in parallel in a single step. A final summation of the results of these computations yield the elements of matrix C in $G0I$. Let $B(G)$ be the time to broadcast on G, and define $B(I)$ similarly. Let $S(H)$ be the time to obtain the sum on H. Then, the completity of the algorithm is $M(GHI) = \max(B(G), B(I)) + S(H) + 1$.

Transitive Closure: Given the $N \times N$ adjacency matrix A for a directed graph, its transitive closure is $A^* = A^N$. We can use the above matrix multiplication algorithm to compute A^* from A in $\log N$ steps by the following algorithm, which assumes that the adjacency matrix is initially stored in the $G0I$ subgraph of GHI. The final result will again be obtained in the $G0I$ subgraph.

```
Algorithm Transitive-Closure($GHI$: Network; $xyz$: Node; $A[xyz]$: Array)
    Repeat for $\log N$ iterations:
            For each $z \in I$ do in parallel:
                Product-Route ($GHz$, $x0z$, $0xz$, $A[x0z]$);
            For each $x \in G$ do in parallel:
                Product-Route ($xHI$, $x0z$, $xz0$, $A[x0z]$);
            Matrix-Multiply ($GHI$, $xyz$, $A[xyz]$, $A[xyz]$, $A[xyz]$)
```

It is also possible to compute other graph theory problems by using the primitive operations provided in this paper. For example, the computation of minimum-weight spanning tree involves point-to-point routing in the factor graph, broadcasting in the factor graph, pointer jumping in a two-dimensional product, and finding the minimum value in a factor graph. While we omit the algorithm due to space limitations, the reader is encouraged to construct such an algorithm on a two-dimensional product network analogous to the basic idea in [7, pp. 325-338].

Fast Fourier Transform: The FFT is a "fast" method to compute the discrete Fourier transform (DFT) of a vector. The DFT of an n-vector u is the n-vector v, defined as

$$v_k = \sum_{j=0}^{n-1} u_j \omega^{kj} \;; \quad k = 0, ..., n-1$$

where ω is a principal nth root of unity (i.e. $\omega^n = 1$ and $\omega^j \neq 0$ for $0 < j < n$). Since the FFT is mostly used for signal processing, it is usually considered to be $\omega = e^{-2\pi i/n}$.

Let $N = L \times M$ and suppose that we have the N-vector stored in the $L \times M$ array in the row-major order. The FFT computation of the N-vector can be decomposed as [10]

$$v_k = v_{s,r} = \sum_{m=0}^{M-1} \omega^{Lmr} \omega^{ms} \sum_{l=0}^{L-1} u_{l,m} \omega^{Msl} \;; \quad where\ k = Lr + s.$$

Therefore we can compute the FFT of the N-vector by first computing the L-point FFT of each column independently, followed by multiplying each term by the *twiddle* factor ω^{ms} for its corresponding m, s values, and then finally computing the FFT of each row independently with w^L as kernel.
Given:

```
Operation FFT($G$: Network; $x$: Node; $r[x], s[x]$: Array)
    Before: $\forall x \in V(G)$, $r[x] = u_x$.
    After: $\forall x \in V(G)$, $s[x] = \sum_{j=0}^{n-1} u_j \omega^{xj}$.
```

Suppose that the elements of the input vector u are initially held in the nodes of GH in row-major order, stored as the array $r[xy]$.

```
Algorithm Product-FFT($GH$: Network; $xy$: Node; $r[xy], s[xy]$: Array)
    For each $y \in V(H)$ do in parallel:
        FFT($Gy$, $r[xy]$, $a[xy]$);
    For each $x \in V(G), y \in V(H)$ do in parallel:
        $b[xy] = a[xy]\omega^{xy}$;
    For each $x \in V(G)$ do in parallel:
        FFT($xH$, $b[xy]$, $s[xy]$);
    Swap dimensions, so that $s[xy] = s[yx]$
```

This last line of the algorithm is necessary to leave the computed vector v in row-major order, since the previous computation obtains it in column-major order. This process does not require any computation, since it is enough to redefine the dimensions as $x \leftarrow y$ and $y \leftarrow x$ to *logically* have the elements at the right nodes. Let $F(G)$ and $F(H)$ be the time to compute the FFT on G and H, respectively. Then, the complexity of this algorithm is $F(GH) = F(G) + F(H) + 1$.

APPLICATION TO SPECIFIC NETWORKS

In this section we obtain the asymptotic complexity of the presented algorithms on several multi-dimensional "homogeneous" product networks. A product network is said to be homogeneous if all its factor networks are isomorphic. We will use N to denote the size of the factor network and r to denote the number of dimensions (number of factor networks) of the product network. The value of r will be 2 or 3 depending on the algorithm.

Grid: The grid is the product of linear arrays. Observe that point-to-point routing, broadcasting, computing sums, and computing the FFT have complexity $O(N)$ on a N-node linear array. Hence, we find that all the presented algorithms, except for the transitive closure, have complexity $O(rN)$. The transitive closure algorithm, has complexity $O(rN \log N)$.

Mesh-connected trees (MCT): This network is the homogeneous product of complete binary trees [2]. On the N-node complete binary tree, point-to-point routing, broadcasting, and computing sums take $O(\log N)$ steps. The FFT operation requires $O(N)$ steps on the complete binary tree since it is restricted by the bisection width of the complete binary tree. For the MCT network, all the presented algorithms have complexity $O(r \log N)$, except the transitive closure algorithm which has complexity $O(r \log^2 N)$, and the FFT algorithm which has complexity $O(rN)$.

Product of de Bruijn (PDB) and shuffle-exchange (PSE) networks: Products of de Bruijn networks were proposed in [3, 11] and products of shuffle-exchange networks were proposed in [3]. In both factor networks, point-to-point routing, broadcasting, computing sums, and computing the FFT have complexity $O(\log N)$ [7]. Then, all the algorithms presented have complexity $O(r \log N)$ on their r-dimensional product except the transitive closure algorithm which has complexity $O(r \log^2 N)$.

Hypercube (HC): The n-dimensional hypercube can be seen as the r-dimensional homogeneous homogeneous product of the i-dimensional hypercube where $n = ri$. Letting $i = logN$, consider

Network	P-p	Broad.	Sum	Pointer J.	Mat. mult.	Trans. Clo.	FFT
Grid	$O(rN)$*	$O(rN)$*	$O(rN)$*	$O(rN)$*	$O(rN)$*	$O(rN\log N)$	$O(rN)$*
MCT	$O(r\log N)$*	$O(r\log N)$*	$O(r\log N)$*	$O(r\log N)$*	$O(r\log N)$*	$O(r\log^2 N)$	$O(rN)$
PDB	$O(r\log N)$*	$O(r\log N)$*	$O(r\log N)$*	$O(r\log N)$*	$O(r\log N)$*	$O(r\log^2 N)$	$O(r\log N)$*
PSE	$O(r\log N)$*	$O(r\log N)$*	$O(r\log N)$*	$O(r\log N)$*	$O(r\log N)$*	$O(r\log^2 N)$	$O(r\log N)$*
HC	$O(r\log N)$*	$O(r\log N)$*	$O(r\log N)$*	$O(rLogN)$*	$O(r\log N)$*	$O(r\log^2 N)$	$O(r\log N)$*

Table 1: Asymptotic running time complexities of the proposed algorithms on several networks. The values marked with "*" are optimal. Note that $r = 2$ for all columns except for matrix multiplication and transitive closure, where $r = 3$.

the running times of primitive low-level operations in this N-node factor network. On the N-node hypercube, point-to-point routing, broadcasting, FFT, and computing sums take $O(\log N)$ steps. Therefore, on the product all the presented algorithms, except for transitive closure, will take $O(r \log N)$ steps. Transitive closure requires $O(r \log^2 N)$ steps.

Table 1 summarizes these running times. The time complexities marked with "*" are optimal for the network considered, since for all of these cases the complexities obtained coincide with the diameters of the corresponding networks. The only exceptions are the transitive closure algorithm, and the FFT on the mesh-connected trees. As shown in [6], a faster algorithm can be obtained for the transitive closure computation and other graph theory problems based on using the pointer jumping algorithm. For FFT on MCT, no algorithm faster than ours is currently available.

CONCLUSIONS

In this paper we have proposed an approach to construct algorithms on product networks that are applicable regardless of the underlying factor network. These algorithms run very efficiently on the example networks explored. This implies that the generality of the approach does not necessarily come at a cost in efficiency. There are still many interesting problems not considered in this paper or in any of the previous papers in the literature. Investigation of new general algorithms for these problems appears to be an interesting area of research.

References

[1] M. Baumslag and F. Annexstein, "A Unified Framework for Off-Line Permutation Routing in Parallel Networks," *Math. Systems Theory*, vol. 24, no. 4, pp. 233–251, 1991.

[2] K. Efe and A. Fernández, "Mesh Connected Trees: A Bridge between Grids and Meshes of Trees," *IEEE Transactions on Parallel and Distributed Systems,* vol. 7, pp. 1283-1293, Dec. 1996.

[3] K. Efe and A. Fernández, "Products of Networks with Logarithmic Diameter and Fixed Degree," *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, pp. 963–975, Sept. 1995.

[4] T. El-Ghazawi and A. Youssef, "A General Framework for Developing Adaptive Fault-Tolerant Routing Algorithms," *IEEE Transactions on Reliability*, vol. 42, pp. 250–258, June 1993.

[5] A. Fernández, N. Eleser, and K. Efe, "Generalized Algorithm for Parallel Sorting on Product Networks," in *Proceedings of the 1995 International Conference on Parallel Processing*, vol. III, pp. 155–159, Aug. 1995.

[6] A. Fernández, “Homogeneous Product Networks for Processor Interconnection,” PhD Thesis, University of Southwestern Louisiana, Dec. 1994.

[7] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, and Hypercubes.* San Mateo: Morgan Kaufmann, 1992.

[8] S. R. Öhring and D. H. Hohndel, “Optimal Fault-Tolerant Communication Algorithms on Product Networks using Spanning Trees,” in *Proceedings of the 6th IEEE Symposium on Parallel and Distributed Processing*, (Dallas,TX), Oct. 1994.

[9] F. Preparata and J. Vuillemin, “Area-optimal VLSI Network for Matrix Multiplication,” in *Proceedings of the 14th Princeton Conference on Information Science and Systems*, pp. 300–309, 1980.

[10] L. R. Rabiner and B. Gold, *Theory and Application of Digital Signal Processing.* Englewood Cliffs, NJ: Prentice-Hall, 1975.

[11] A. L. Rosenberg, “Product-Shuffle Networks: Toward Reconciling Shuffles and Butterflies,” *Discrete Applied Mathematics*, vol. 37/38, pp. 465–488, July 1992.

[12] A. Youssef, “Cartesian Product Networks,” in *Proceedings of the 1991 International Conference on Parallel Processing*, vol. I, pp. 684–685, Aug. 1991.