QUOTIENT NETWORKS

by

John P. Fishburn
Raphael A. Finkel

Computer Sciences Technical Report #419

February 1981

Quotient Networks

by

John P. Fishburn

Raphael A. Finkel

Computer Sciences Department

University of Wisconsin - Madison

Abstract

A _large-network algorithm_ solves a problem of size N on a network of N processors. We present a general method for transforming large-network algorithms into _quotient-network algorithms_, which solve problems of size N on networks with fewer processors. This transformation allows algorithms to be designed assuming any number of processing elements. The implementation of such algorithms on a quotient network results in no loss of efficiency, and often a great savings in hardware cost.

TABLE OF CONTENTS

1. INTRODUCTION.................................. 2

2. EXISTING NETWORKS............................. 3

2.1. Grid-Connected Network................. 3
2.2. Perfect Shuffle........................ 4
2.3. PM2I................................... 4
2.4. Cube................................... 5

3. EXISTING ALGORITHMS........................... 5

3.1. Fast-Fourier Transform on the Shuffle.. 5
3.2. Sorting on the Shuffle................. 9
3.3. Polynomial Evaluation on the Shuffle... 9
3.4. Finite-difference Methods.............. 10

4. NETWORK EMULATION............................. 10

4.1. Perfect Shuffle........................ 12
4.2. Grid-connected Network................. 15
4.3. Cube................................... 18
4.4. PM2I................................... 20

5. SOME RESULTING ALGORITHMS..................... 22

5.1. Fast-Fourier Transform on the Shuffle.. 22
5.2. Sorting on the Shuffle................. 24
5.3. Polynomial Evaluation on the Shuffle... 25
5.4. Finite-difference Methods.............. 25

6. THE ECONOMICS OF EMULATION.................... 26

7. CONCLUSIONS................................... 27

8. ACKNOWLEDGEMENTS.............................. 28

REFERENCES...................................... 29

1. INTRODUCTION

One barrier to the practical use of interconnection networks is the lack of algorithms for processing large problems on small machines. (By an interconnection network we mean an SIMD parallel computer interconnected by some interconnection strategy.) Typically, it is assumed that N processors are available to process N data [1,2]. If we happen to have N+1 or more data points, then we must choose between the serial algorithm and a bigger machine. Notable exceptions can be found in work by Baudet and Stevenson [3], and by Siegel, Mueller and Siegel [4]. This paper investigates a method that constructs algorithms for solving large problems on small networks. We call these algorithms _quotient-network algorithms_. In Section 2, we review some proposed interconnection networks. Section 3 reviews proposed algorithms for those networks. We call these algorithms _large-network algorithms_, since each one assumes as many processors as points in the problem to be solved. Section 4 presents a general method for transforming a large-network algorithm into a quotient-network algorithm. Section 5 applies this method to each of the algorithm-machine combinations of Section 3. Section 6 discusses some economic advantages of quotient-network algorithms.

2. EXISTING NETWORKS

In this section we briefly review some proposed interconnection networks. For a more thorough overview, see [5]. We assume that each network contains N processors. We denote the square root of N by n, and $\log_2 N$ by m. We will name the processors PE(Ø) through PE(N-1). Sometimes we refer to a processor by the binary form of its number, $p = p_{m-1}p_{m-2}\cdots p_1 p_\emptyset$.

2.1. Grid-Connected Network

In this network, the processors are arranged in a two-dimensional n by n grid. The processor in the ith row and jth column is named PE(i,j), for $\emptyset \leq i,j < n$. A processor is connected to its north, south, east and west neighbors:

```
If i > Ø,   PE(i,j) is connected to PE(i-1,j).
If i < n-1, PE(i,j) is connected to PE(i+1,j).
If j > Ø,   PE(i,j) is connected to PE(i,j-1).
If j < n-1, PE(i,j) is connected to PE(i,j+1).
```

The Illiac IV network adds additional connections between edge processors [6].

2.2. Perfect Shuffle

Shuffle-Exchange

In this network, $PE(p_{m-1}p_{m-2}\cdots p_1p_\emptyset)$ is connected to $PE(p_{m-2}\cdots p_1p_\emptyset p_{m-1})$ by the shuffle function and to $PE(p_{m-1}p_{m-2}\cdots p_1\bar{p}_\emptyset)$ by the exchange function.

4-Pin Shuffle

In this network, each processor has two input pins IPINØ and IPIN1, and two output pins OPINØ and OPIN1. We can number all input pins by assigning to IPINØ on processor $p_{m-1}p_{m-2}\cdots p_1p_\emptyset$ the number $p_{m-1}p_{m-2}\cdots p_1p_\emptyset\emptyset$. IPIN1 on the same processor is assigned the number $p_{m-1}p_{m-2}\cdots p_1p_\emptyset 1$. This numbering allows us to refer to input pins as IPIN(Ø) through IPIN(2N-1). Output pins are numbered in the same way, OPIN(Ø) through OPIN(2N-1). The shuffle function is used to transfer data from the output pins of $PE(p_{m-1}p_{m-2}\cdots p_1p_\emptyset)$ to the input pins of processors $PE(p_{m-2}\cdots p_1p_\emptyset\emptyset)$ and $PE(p_{m-2}\cdots p_1p_\emptyset 1)$.

2.3. PM2I

In the Plus-Minus 2^i network (PM2I), PE(j) is connected to processors

$j+2^i \bmod N$

and

$j-2^i \bmod N$,

for $\emptyset \leq i < m$.

2.4. Cube

$PE(p_{m-1} \cdots p_{i+1} p_i p_{i-1} \cdots p_\emptyset)$ in the cube network is connected to the m processors $PE(p_{m-1} \cdots p_{i+1} \bar{p}_i p_{i-1} \cdots p_\emptyset)$, for $\emptyset \leq i < m$.

3. EXISTING ALGORITHMS

In this section we review some proposed large-network algorithms. The number of such algorithms is large and growing, so we do not attempt to be comprehensive. Our goal is to illustrate the process of transforming large-network algorithms into quotient-network algorithms.

3.1. Fast-Fourier Transform on the Shuffle

Let $A(k)$, $k=\emptyset, 1, \ldots, N-1$, be a vector of N complex numbers. The *Discrete Fourier Transform* (DFT) of A is defined to be the vector

(1) $$X(j) = \sum_{k=\emptyset}^{N-1} A(k)W^{jk} \qquad j=\emptyset,1,\ldots,N-1$$

where $W = e^{2\pi i/N}$. The obvious algorithm for computing X takes time $O(N^2)$. An important advance in the theory of algorithms was the discovery of an O(Nlog N) algorithm for the DFT [7]. This algorithm is called the <u>Fast Fourier Transform</u> (FFT). Pease [8] has discovered an algorithm that computes the FFT on N/2 processors in time proportional to log N, thus achieving optimal speedup. Pease's algorithm can be explained as follows: First, we represent both k and j by their binary expansion.

$$k = k_{m-1}k_{m-2}\cdots k_\emptyset$$

and

$$j = j_{m-1}j_{m-2}\cdots j_\emptyset.$$

Equation 1 then becomes

(2) $X(j)$

$$=\sum_{k_\emptyset}\sum_{k_1}\cdots\sum_{k_{m-1}} A(k_{m-1}k_{m-2}\cdots k_\emptyset)W^{jk_{m-1}2^{m-1}}W^{jk_{m-2}2^{m-2}}\cdots W^{jk_\emptyset}$$

$$=\sum_{k_\emptyset} W^{jk_\emptyset}\sum_{k_1} W^{jk_1 2}\cdots\sum_{k_{m-1}} W^{jk_{m-1}2^{m-1}} A(k_{m-1}k_{m-2}\cdots k_\emptyset).$$

Equation 2 consists of m nested summations. Since $W^N=1$,

$$W^{j2^{m-s}} = W^{j_{s-1}j_{s-2}\cdots j_\emptyset \cdot 2^{m-s}},$$

so the innermost s summations depend only on the m binary variables $j_\emptyset, \ldots, j_{s-1}$ and $k_{m-s-1}, \ldots, k_\emptyset$. Thus the innermost s summations represent a function from $\emptyset, \ldots, N-1$ to the complex numbers; we represent this function as an array B_s of N complex numbers. B_s satisfies

$$B_\emptyset(k_{m-1}k_{m-2}\cdots k_\emptyset) = A(k_{m-1}k_{m-2}\cdots k_\emptyset),$$

(3) $$B_s(j_\emptyset\cdots j_{s-1}k_{m-s-1}\cdots k_\emptyset) =$$

$$\sum_{k_{m-s}} B_{s-1}(j_\emptyset\cdots j_{s-2}k_{m-s}\cdots k_\emptyset)W^{j_{s-1}j_{s-2}\cdots j_\emptyset \cdot 2^{m-s} \cdot k_{m-s}},$$

and

$$B_m(j_\emptyset\cdots j_{m-1}) = X(j_{m-1}\cdots j_\emptyset).$$

Equation 3 reveals how we can use the 4-pin shuffle to compute the FFT: We iteratively compute B_s for s = 1 to m. Iteration s results in B_s distributed on the output pins. To perform iteration s, we form the weighted sum of elements from B_{s-1} whose indices differ only in bit position number m-s. The 4-pin shuffle with N/2 processors provides exactly the data alignment we want, since shuffling an array s times causes the indices of the two numbers in each processor to differ only in bit position number m-s.

The following is a description of Pease's parallel FFT algorithm. The hardware is assumed to be a 4-pin shuffle with N/2 PEs. The machine operates in SIMD mode, and PEs differ only in that each processor $PE(p_{m-2}\cdots p_\emptyset)$ knows its own address $p_{m-2}\cdots p_\emptyset$.

Large-network Parallel FFT

```
Input: data items $A(k)$  $k=0, \ldots, N-1$
     with $A(k)$ on $OPIN(k)$

Output: the Fourier transform $X(j)$ of $A(k)$
     with $X(j_{m-1} \ldots j_0)$ on $OPIN(j_0 \ldots j_{m-1})$

for s := 1 to m
begin
     SHUFFLE;
     $OPIN0 := IPIN0 + W^{0p_0 \ldots p_{s-2} \cdot 2^{m-s}} \cdot IPIN1$;
     $OPIN1 := IPIN0 + W^{1p_0 \ldots p_{s-2} \cdot 2^{m-s}} \cdot IPIN1$;
end
```

This algorithm can be proved correct by induction on the following loop invariant:

Immediately after shuffle number s,

$B_{s-1}(j_0 \ldots j_{s-2} 0 k_{m-s-1} \ldots k_0)$

and

$B_{s-1}(j_0 \ldots j_{s-2} 1 k_{m-s-1} \ldots k_0)$

are in processor $PE(k_{m-s-1} \ldots k_0 j_0 \ldots j_{s-2})$ at pin positions

$IPIN(k_{m-s-1} \ldots k_0 j_0 \ldots j_{s-2} 0)$

and

$IPIN(k_{m-s-1} \ldots k_0 j_0 \ldots j_{s-2} 1)$,

respectively. This processor then places

$B_s(j_0 \ldots j_{s-2} 0 k_{m-s-1} \ldots k_0)$

and

$B_s(j_0 \ldots j_{s-2} 1 k_{m-s-1} \ldots k_0)$

onto output pin positions

and $\text{OPIN}(k_{m-s-1} \ldots k_{\emptyset} j_{\emptyset} \ldots j_{s-2} \emptyset)$

$\text{OPIN}(k_{m-s-1} \ldots k_{\emptyset} j_{\emptyset} \ldots j_{s-2} 1)$,

respectively.

3.2. Sorting on the Shuffle

Batcher's algorithm [9], as adapted by Stone [2], sorts N numbers in $\log^2 N$ passes through the N/2-processor 4-pin shuffle. After each shuffle, a processor either

1. Copies the two inputs directly to the two outputs.
2. Compares the two inputs and puts the lower on OPINØ and the higher on OPIN1.
3. Compares the two inputs and puts the higher on OPINØ and the lower on OPIN1.

Hence Batcher's algorithm requires $\log^2 N$ shuffle steps on the 4-pin shuffle.

3.3. Polynomial Evaluation on the Shuffle

Consider the problem of evaluating the (N-2)nd-degree polynomial

$$\sum_{i=\emptyset}^{N-2} a_i x^i \tag{4}$$

for given numbers x and a_i, $i=\emptyset, \ldots, N-2$. Horner's rule, which evaluates a polynomial by the scheme

$(\ldots((a_n x+a_{n-1})x+a_{n-2})x+\ldots+a_1)x+a_0$,

is an optimal serial algorithm that requires exactly N-2 multiplications and N-2 additions. Stone [2] presents an algorithm for computing (4) with 2 log N passes through the N/2-processor 4-pin shuffle.

3.4. Finite-difference Methods

The literature is full of proposals for the parallel execution of finite-difference calculations [10,11,12,13,14,15]. Often, the rectilinear problem grid is mapped one-to-one onto the rectilinear processor grid. At each time step, each processor communicates its values to and receives values from each of its nearest neighbors. This exchange provides each processor with the necessary values to compute the value of its point at the next time step.

4. NETWORK EMULATION

Definition: Suppose that $G = (V_G,E_G)$ and $H = (V_H,E_H)$ are graphs. We say that a function $f:V_H \dashrightarrow V_G$ is an emulation of H by G if for every edge $(h_1,h_2) \in E_H$

$f(h_1) = f(h_2)$ or $(f(h_1), f(h_2)) \in E_G$.

Every emulation $f: V_H \dashrightarrow V_G$ induces a mapping $f': E_H \dashrightarrow V_G \sqcup E_G$ in a natural way:

$f'(h_1,h_2) = (f(h_1), f(h_2))$ if $(f(h_1), f(h_2)) \in E_G$

otherwise

$f'(h_1,h_2) = f(h_1) = f(h_2)$.

We say that the node $g \in V_G$ <u>emulates</u> the nodes $f^{-1}(g)$, and that the edge $(g_1,g_2) \in E_G$ <u>emulates</u> the edges $f'^{-1}(g_1,g_2)$. If $|f^{-1}(g)|$ is the same for every $g \in V_G$, then we say that f is <u>computationally uniform</u>, and $|f^{-1}(g)|$ is the <u>computation factor</u> of f; if $|f'^{-1}(e)|$ is the same for every $e \in E_G$, then we say that f is <u>exchange-uniform</u>, and $|f'^{-1}(e)|$ is the <u>exchange factor</u> of f. If f is computationally uniform and exchange-uniform, and if the computation factor equals the exchange factor, then we say that f is <u>totally uniform</u>, and $|f^{-1}(g)|$ is the <u>emulation factor</u> of f.

If the graphs G and H are interconnection networks, then the existence of an emulation of H by G provides a way for the network G to emulate the actions of the network H. By analogy with the notion of quotient groups in abstract algebra, we call G a <u>quotient network</u>. The processor $g \in V_G$ is time-shared to emulate the group of processors $f^{-1}(g)$ in V_H, and the communications line $(g_1,g_2) \in E_G$ is time-multiplexed to emulate the communication lines $f'^{-1}(g_1,g_2)$ in E_H.

If the emulation of H by G is computationally uniform, then the processors in G can efficiently perform the actions of the processors of H: Since each processor in G emulates the same number of processors of H, all of the processors in G can proceed in unison and finish simultaneously. No processors sit idle while other overloaded processors finish their work. Likewise, if the emulation of H by G is exchange-uniform, then the communications lines in G can efficiently perform the actions of the communications lines of H: Since each communications line in G emulates the same number of communications lines of H, all of the data transfers in G can proceed in unison and finish simultaneously. No communications lines sit idle while other overloaded communications lines finish their work.

We now present an emulation for each of the networks reviewed in Section 2. In each case, a large network H is emulated by a smaller network G of the same general interconnection scheme.

4.1. Perfect Shuffle

Suppose that H is a shuffle-exchange network with $N = 2^m$ processors. We will emulate this network with a 4-pin shuffle network of size N/2, and then emulate the 4-pin shuffle network with any 4-pin shuffle network of size a smaller power of two.

<u>Theorem 1</u>: The function $f(p_{m-1}\cdots p_2p_1p_\emptyset) = p_{m-1}\cdots p_2p_1$ emulates the shuffle-exchange network of size N with the 4-pin shuffle of size N/2.

<u>Proof</u>: Suppose that $e = (h_1,h_2) \in E_H$. If e is an exchange connection, then $f(h_1) = f(h_2)$. If e is a shuffle connection, then

$$\begin{aligned}
&(f(h_1),f(h_2))\\
&= (f(p_{m-1}\cdots p_1p_\emptyset),f(p_{m-2}\cdots p_1p_\emptyset p_{m-1}))\\
&= (p_{m-1}\cdots p_1,p_{m-2}\cdots p_1p_\emptyset) \in E_G.
\end{aligned}$$

Q.E.D.

The emulation f is computationally uniform and exchange-uniform, but not totally uniform. The computation factor is two and the exchange factor is one.

<u>Theorem 2</u>: The function

$$f(p_{m+q-1}p_{m+q-2}\cdots p_qp_{q-1}\cdots p_\emptyset) = p_{m+q-1}p_{m+q-2}\cdots p_q$$

emulates the 4-pin shuffle of size $NP=2^{m+q}$ with the 4-pin shuffle of size $N=2^m$.

<u>Proof</u>: Let

$$\begin{aligned}
&(h_1,h_2)\\
&= (p_{m+q-1}p_{m+q-2}\cdots p_qp_{q-1}\cdots p_\emptyset,\ p_{m+q-2}\cdots p_qp_{q-1}\cdots p_\emptyset X)
\end{aligned}$$

be an edge in H. Then

$$(f(h_1),f(h_2))$$
$$= (p_{m+q-1}p_{m+q-2}\cdots p_q,\ p_{m+q-2}\cdots p_q p_{q-1}) \in E_G.$$

Q.E.D.

The emulation f is totally uniform, with emulation factor 2^q. Figure 1 illustrates a 4-pin shuffle with four PEs emulating a 4-pin shuffle with eight PEs.

4.2. Grid-connected Network

The emulation of a large grid-connected network with a small one is fairly straightforward; we simply partition the large network into square regions.

Theorem 3: The function

$$f(p_{r+s-1}\cdots p_r p_{r-1}\cdots p_\emptyset,\ q_{r+s-1}\cdots q_r q_{r-1}\cdots q_\emptyset)$$
$$= (p_{r+s-1}\cdots p_r,\ q_{r+s-1}\cdots q_r)$$

is an emulation of a grid-connected network of size 2^{2r+2s} by a grid-connected network of size 2^{2s}.

Proof: Suppose that $h = (h_1,h_2) = ((P_1,Q_1),(P_2,Q_2)) \in E_H$. We assume that h is a "North" connection, so that $P_1 = P_2$ and $Q_1 = Q_2 - 1$. A similar proof can be used when h is any of the other three grid connections. We can represent P_1, P_2, Q_1, and Q_2 as follows:

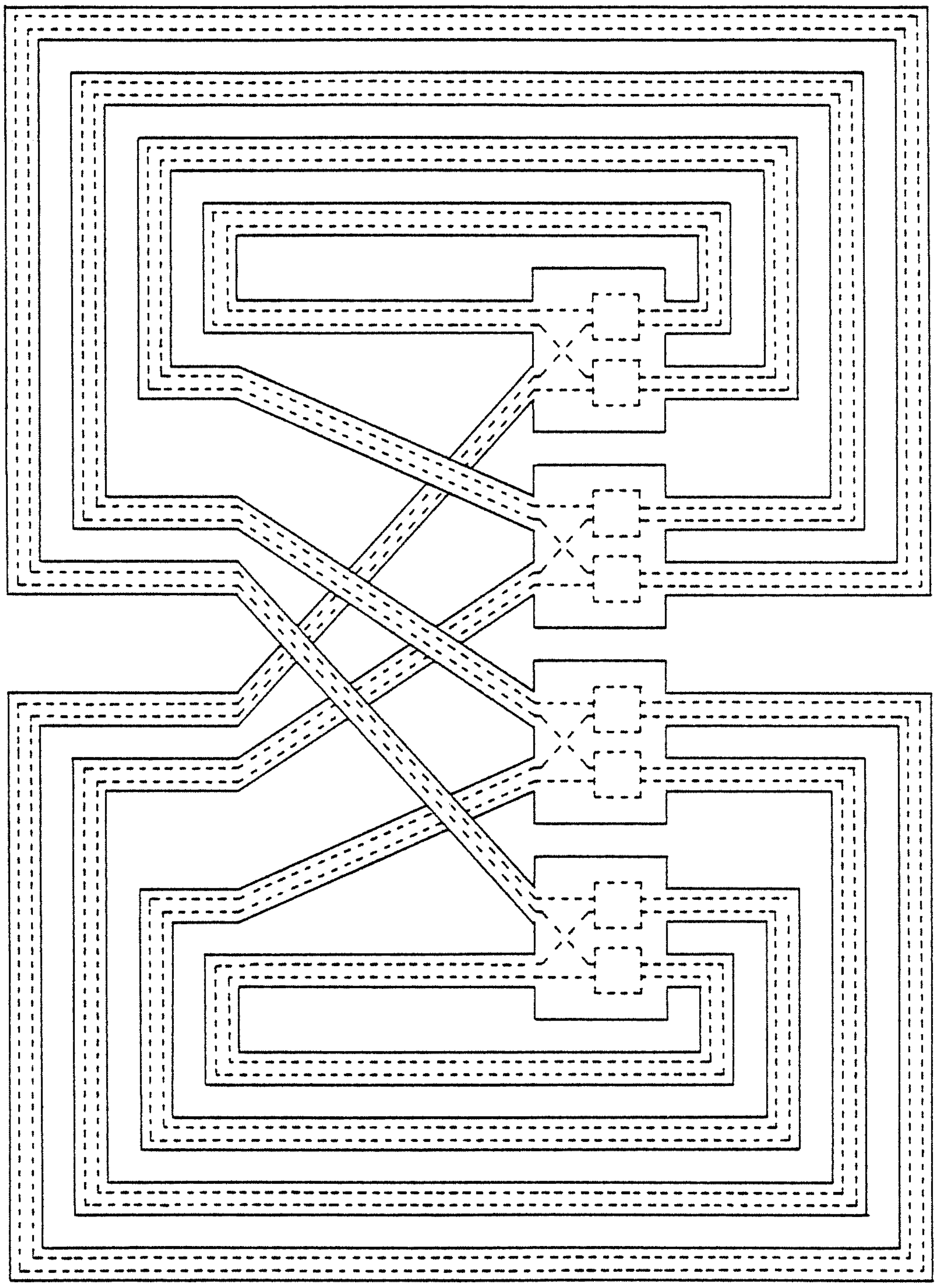

Fig. 1. 4-PE 4-pin shuffle emulating 8-PE 4-pin shuffle.

$$P_1 = P_2 = p_{r+s-1} \cdots p_r p_{r-1} \cdots p_\emptyset$$

$$Q_1 = Q_2 - 1 = q_{r+s-1} \cdots q_r q_{r-1} \cdots q_\emptyset$$

If incrementing Q_1 results in a carry into the top s bits in its binary representation, then

$$\begin{aligned}
&(f(h_1), f(h_2)) \\
&= (f(p_{r+s-1} \cdots p_r p_{r-1} \cdots p_\emptyset,\ q_{r+s-1} \cdots q_r q_{r-1} \cdots q_\emptyset), \\
&\qquad f(p_{r+s-1} \cdots p_r p_{r-1} \cdots p_\emptyset,\ q_{r+s-1} \cdots q_r q_{r-1} \cdots q_\emptyset + 1)) \\
&= ((p_{r+s-1} \cdots p_r,\ q_{r+s-1} \cdots q_r), \\
&\qquad (p_{r+s-1} \cdots p_r,\ q_{r+s-1} \cdots q_r + 1)) \\
&\in E_G.
\end{aligned}$$

If not, then

$$\begin{aligned}
&f(h_1) \\
&= f(P_1, Q_1) \\
&= f(p_{r+s-1} \cdots p_r p_{r-1} \cdots p_\emptyset,\ q_{r+s-1} \cdots q_r q_{r-1} \cdots q_\emptyset) \\
&= (p_{r+s-1} \cdots p_r,\ q_{r+s-1} \cdots q_r) \\
&= f(p_{r+s-1} \cdots p_r p_{r-1} \cdots p_\emptyset,\ q_{r+s-1} \cdots q_r q_{r-1} \cdots q_\emptyset + 1) \\
&= f(P_2, Q_2) \\
&= f(h_2).
\end{aligned}$$

Q.E.D.

The emulation f is computationally uniform and exchange-uniform, but not totally uniform. The computation factor is 2^{2r} and the exchange factor is 2^r. Figure 2 illustrates part of a grid-connected network emulating a grid-connected network that is four times larger.

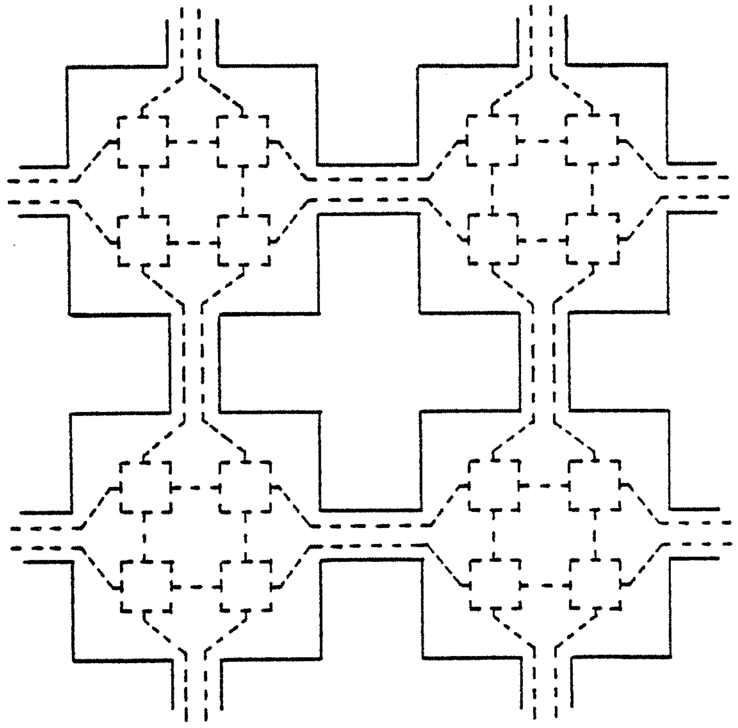

Fig. 2. Grid emulation with r=2.

In general, a k-dimensional grid may be emulated by a smaller k-dimensional grid. For a given r, the exchange factor is 2^r and the computation factor is 2^{kr}.

4.3. Cube

Theorem 4: The function

$$f(p_{m+q-1}p_{m+q-2}\cdots p_q p_{q-1}\cdots p_\emptyset) = p_{m+q-1}p_{m+q-2}\cdots p_q$$

emulates the cube of size $NP=2^{m+q}$ with the cube of size $N=2^m$.

Proof: Suppose that $(h_1,h_2) \in E_H$. Then h_1 and h_2 are of the form

$$h_1 = p_{m+q-1}\cdots p_i\cdots p_\emptyset$$
and
$$h_2 = p_{m+q-1}\cdots \bar{p}_i\cdots p_\emptyset.$$

If $i \leq q$, then $f(h_1) = f(h_2)$. If $i > q$, then

$$f(h_1) = p_{m+q-1}\cdots p_i\cdots p_q$$
and
$$f(h_2) = p_{m+q-1}\cdots \bar{p}_i\cdots p_q.$$

Hence $(f(h_1),f(h_2)) \in E_G$.

Q.E.D.

The emulation f is totally uniform, with emulation factor 2^q. Any function that discards q bits and permutes the remaining bits is also an emulation. Figure 3 illustrates a cube of size four emulating a cube of size eight.

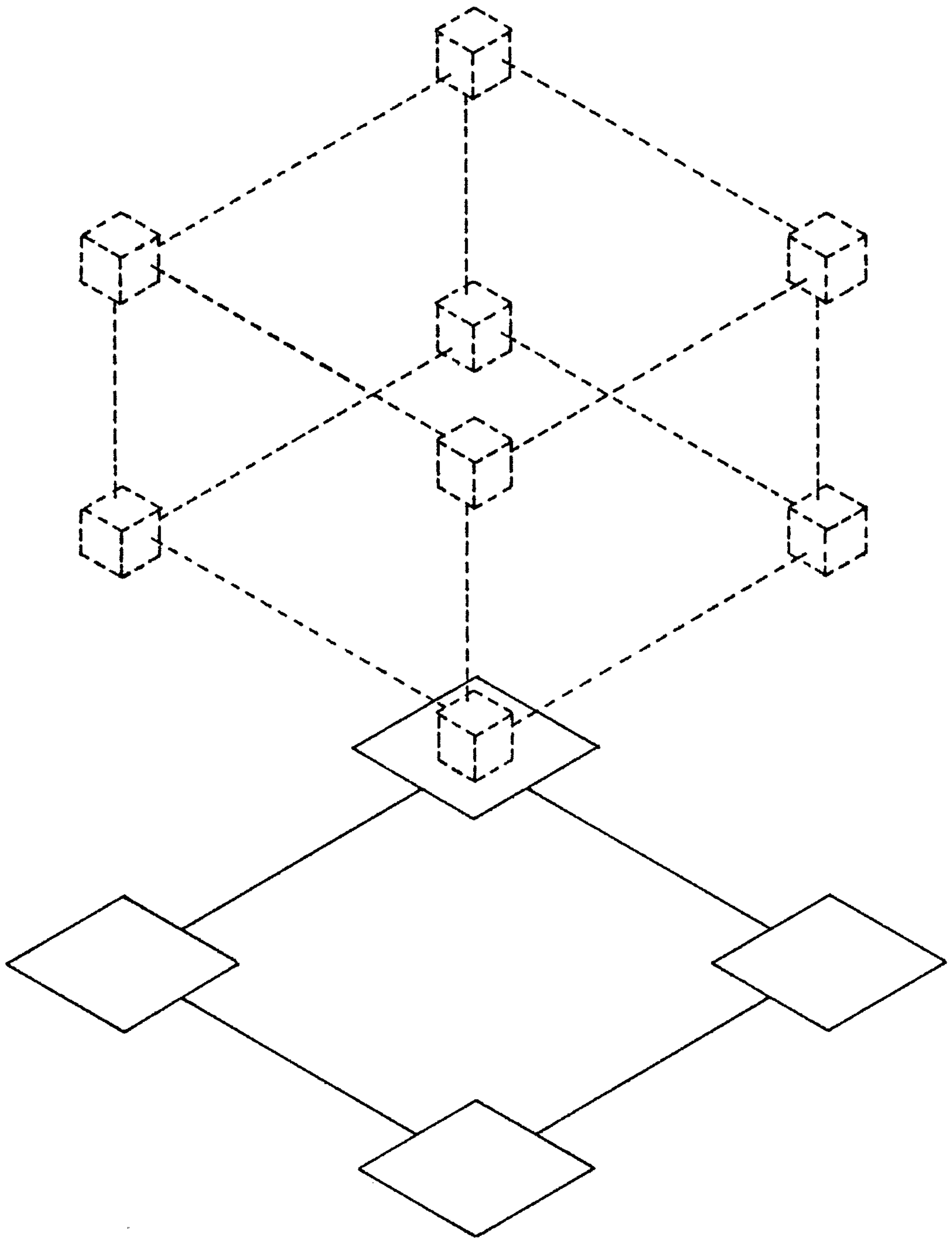

Fig. 3. Two-dimensional cube emulating three-dimensional cube.

4.4. PM2I

Theorem 5: The function

$$f(p_{m+q-1}p_{m+q-2}\cdots p_q p_{q-1}\cdots p_0) = p_{m+q-1}p_{m+q-2}\cdots p_q$$

emulates the PM2I network of size $NP=2^{m+q}$ with the PM2I network of size $N=2^m$.

Proof: Let $(h_1,h_2) \in E_H$. Hence h_1 and h_2 are of the form

$$\text{and}\quad \begin{array}{l} h_1 = p_{m+q-1}p_{m+q-2}\cdots p_q p_{q-1}\cdots p_0 \\ h_2 = p_{m+q-1}p_{m+q-2}\cdots p_q p_{q-1}\cdots p_0 + 2^i, \end{array}$$

for some $0 \le i < m$. If $i < q$ and if the addition of 2^i to h_1 does not cause a carry into the top m bits of its address, then $f(h_1) = f(h_2)$. Otherwise, if $i \ge q$ then

$$f(h_2) = f(h_1) + 2^{i-q},$$

and if $i < q$ then

$$f(h_2) = f(h_1) + 1.$$

In either case, $(f(h_1),f(h_2)) \in E_G$.

Q.E.D.

The emulation f is computationally uniform, with computation factor 2^q. But f is not exchange-uniform, since each "+1" link in G emulates $2^{m+1}-1$ links in H, while every other link in G emulates 2^m links. Figure 4 illustrates a PM2I of size eight emulating a PM2I of size sixteen.

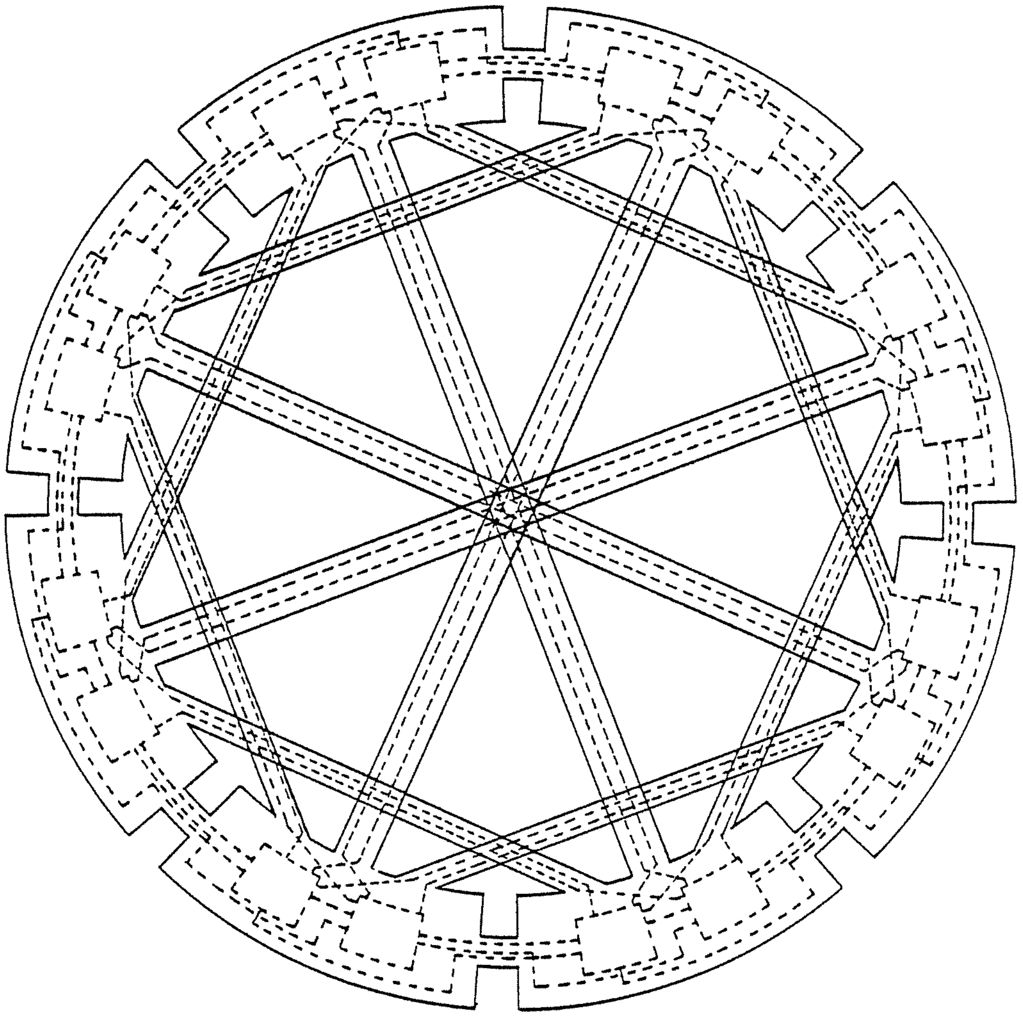

Fig. 4. PM2I of size eight emulating PM2I of size sixteen.

5. SOME RESULTING ALGORITHMS

In this section we transform the large-network algorithms of Section 3 into quotient-network algorithms. Since the transformation is a fairly mechanical one, we present it in detail only for the FFT algorithm. For the other algorithms we only summarize the result.

5.1. Fast-Fourier Transform on the Shuffle

The FFT algorithm presented in Section 3 consists of a loop executed m times. The body of the loop consists of a SHUFFLE followed by placing weighted sums of the input pins onto the output pins. We assume, as in Section 3, that our network contains $N/2 = 2^{m-1}$ machines. We wish to compute the DFT of $NP = 2^{m+q}$ data items A(i) for i=Ø, ..., NP-1. We therefore emulate the actions of a 4-pin network of size $NP/2 = 2^{m+q-1}$. Each processor represents the virtual pins of the processors it is emulating by arrays: The virtual $PE(k_{m+q-2}\cdots k_{\emptyset})$ has pins OPINØ, OPIN1, IPINØ, and IPIN1. These pins are emulated on actual $PE(k_{m+q-2}\cdots k_{q})$ at index $k_{q-1}\cdots k_{\emptyset}$ of arrays EOPINØ, EOPIN1, EIPINØ, and EIPIN1, respectively. Here is the quotient-network FFT algorithm:

Quotient-Network Parallel FFT

```
Input:  data items A(i), i=Ø, ..., NP-1 such that

     $A(k_{m+q-1}\ldots k_{q+1}k_q\ldots k_Ø)$ is stored on machine

     $PE(k_{m+q-1}\ldots k_{q+1})$ in $EOPINk_Ø[k_q\ldots k_1]$

Output: The Discrete Fourier Transform X(i),
     i=Ø, ..., NP-1 such that

     $X(k_Ø\ldots k_q k_{q+1}\ldots k_{m+q-1})$ is stored on machine

     $PE(k_{m+q-1}\ldots k_{q+1})$ in $EOPINk_Ø[k_q\ldots k_1]$

for s := 1 to m do
begin

     { emulate SHUFFLE: }

     for j := Ø to $2^q$ -1 do
     begin { emulate $PE(P_{m-2}\ldots P_Ø j_{q-1}\ldots j_Ø)$ }
          if j is even then
          begin
               OPINØ := EOPINØ[j/2];
               OPIN1 := EOPINØ[$j/2+2^{q-1}$];
          end else
          begin
               OPINØ := EOPIN1[$\lfloor j/2 \rfloor$];
               OPIN1 := EOPIN1[$\lfloor j/2 \rfloor+2^{q-1}$];
          end;
          SHUFFLE;
          EIPINØ[j] := IPINØ;
          EIPIN1[j] := IPIN1;
     end;

     { emulate computation: }
     for j := Ø to $2^q$ -1 do
     begin { emulate $PE(P_{m-2}\ldots P_Ø j_{q-1}\ldots j_Ø)$ }

          $e_{m+q-2}\ldots e_Ø$ := $P_{m-2}\ldots P_Ø j_{q-1}\ldots j_Ø$;

          EOPINØ[j] :=

               EIPINØ[j] + $W^{Øe_Ø\ldots e_{s-2}\cdot 2^{m-s}}$ ·EIPIN1[j];

          EOPIN1[j] :=

               EIPINØ[j] + $W^{1e_Ø\ldots e_{s-2}\cdot 2^{m-s}}$ ·EIPIN1[j];
```

```
        end;
    end;
```

The original large-network FFT algorithm is optimal in the number of processors; that is, the speedup is proportional to the number of processors used. In the above example, the large-network FFT algorithm rests on top of an emulation, which rests on the actual hardware. Apart from the loop and indexing overhead needed to emulate the SHUFFLE step, the 2^{m-1}-processor network is 2^q times as slow as the 2^{m+q-1}-processor network. The loop and indexing overhead slows the algorithm down by only a constant factor, and could be eliminated entirely by unrolling the loops. Therefore the quotient-network algorithm is also optimal: We gain approximately N speedup with N processors. While the original large-network FFT algorithm performs log N operate-shuffle steps, the quotient-network algorithm performs $2^q \log N$ operate-shuffle steps.

5.2. Sorting on the Shuffle

As we have seen, Batcher's large-network algorithm sorts $N = 2^{m+q}$ numbers in $(m+q)^2$ operate-shuffle steps on a 4-pin shuffle with 2^{m+q-1} processors. A quotient-network version of this algorithm sorts the 2^{m+q} numbers in $2^q(m+q)^2$ operate-shuffle steps on a 4-pin shuffle with 2^{m-1} processors.

5.3. Polynomial Evaluation on the Shuffle

A 2^{m+q-1}-processor 4-pin shuffle network can evaluate a polynomial of degree $2^{m+q-1}-2$ in $2(m+q)$ operate-shuffle steps. The quotient-network version of this algorithm evaluates the same polynomial in $2^{q+1}(m+q)$ operate-shuffle steps with 2^{m-1} processors.

5.4. Finite-difference Methods

A large-network algorithm that maps the 2^{2r+2s} points of a finite-difference grid one-to-one onto a 2^{2r+2s} grid-connected network must communicate each point at each time step to all four neighbors. The quotient-network version of this algorithm reduces the communication/computation ratio by communicating only the border points of a processor's region to that processor's neighbors. For a detailed discussion of quotient-network finite-difference methods, see [16].

6. THE ECONOMICS OF EMULATION

We can give several economic arguments in favor of solving large problems on small networks through emulation.

1. The cost of a word of storage is much smaller than the cost of a processor. This fact is independent of further increases in scale of integration. By adding extra storage at each processor in G, we increase the potential computation factor of an emulation; that is, we can emulate a larger H. We therefore increase the largest problem that the network can handle, with a much smaller increase in hardware cost than would be incurred by expanding G to H.
2. Suppose that a solution must meet a time constraint for a problem of size N. One processor cannot meet this constraint, but N processors (the network H) are much too expensive and much faster than needed. An intermediate number of processors (the network G) emulating H may be fast enough and affordable.
3. Given a large-network algorithm, an emulation automatically produces a quotient-network algorithm to solve the same problem on a smaller machine. We achieve economy of thought by solving once and for all the "emulation problem": doing on a small machine what a large machine can do. Thereafter, we can deal exclusively with the simpler class of problems: data

sets of size N on networks with N processors.

7. CONCLUSIONS

We have defined the concept of emulation of one interconnection network by a quotient network. We have seen that quotient networks of various intermediate sizes may be formed for the shuffle-exchange, the 4-pin shuffle, the cube, and the PM2I. These emulations can be evaluated on the basis of computational and exchange uniformity. We plan to extend this work to other networks. Preliminary results imply that the Lens [17], for example, can be emulated, but not in a computationally uniform manner.

Emulations lead to a general method for transforming large-network algorithms into quotient-network algorithms, which solve problems of size N on networks with fewer processors. We have carried out this method in detail for the FFT; similar transformations are possible for a large number of SIMD algorithms that depend on a particular interconnection scheme.

This transformation allows algorithms to be designed assuming any number of processing elements. The implementation of such algorithms on a quotient network results in no loss of efficiency, and often a great savings in hardware cost.

8. ACKNOWLEDGEMENTS

We are grateful to S. Diane Smith for her careful reading of the paper. We have also benefitted from many discussions with the Arachne group: Jonathan Kepecs, Will Leland, M. V. S. Ramanath, Marvin Solomon, and Sun Zhong Xiu.

REFERENCES

[1] J. L. Bentley and H. T. Kung, "A tree machine for searching problems," Proc. 1979 International Conference on Parallel Processing, pp. 257-266 (August 1979).

[2] H. S. Stone, "Parallel processing with the perfect shuffle," IEEE Transactions on Computers C-2Ø, 2, pp. 153-161 (February 1971).

[3] G. M. Baudet and D. Stevenson, "Optimal Sorting Algorithms for Parallel Computers," IEEE Transactions on Computers C-27, 1, pp. 84-87 (January 1978).

[4] L. J. Siegel, P. T. Mueller, and H. J. Siegel, "FFT algorithms for SIMD machines," Proc. Allerton Conference on Communication, Control, and Computing, pp. 1ØØ6-1Ø15 (October 1979).

[5] H. J. Siegel, "Analysis techniques for SIMD machine interconnection networks and the effects of processor address masks," IEEE Transactions on Computers C-26, 2, pp. 153-161 (February 1977).

[6] W. J. Bouknight et al., "The Illiac IV system," Proc. IEEE 6Ø, 4, pp. 369-388 (April 1972).

[7] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," Math. Comput. 19, pp. 297-3Ø1 (April 1965).

[8] M. C. Pease, "An adaptation of the fast Fourier Transform for parallel processing," Journal of the ACM 15, pp. 252-264 (April 1968).

[9] K. E. Batcher, "Sorting networks and their applications," Proc. Spring Joint Comput. Conf. 32, pp. 3Ø7-314 (1968).

[10] H. S. Stone, "Problems of Parallel Computation," Complexity of Sequential and Parallel Numerical Algorithms, Academic Press, (1973).

[11] J. L. Rosenfeld, "A case study in programming for parallel-processors," CACM 12, 12, pp. 645-655 (December 1969).

[12] H. O. Welch, "Numerical weather prediction in the Pepe parallel processor," Proc. 1977 International Conference on Parallel Processing, pp. 186-192 (August 1977).

[13] P. M. Flanders, D. J. Hunt, S. F. Reddaway, and D. Parkinson, "Efficient high speed computing with the Distributed Array Processor," Symposium on High Speed Computer and Algorithm Organization, pp. 113-128 (1977).

[14] D. Chazan and W. Mirankar, "Chaotic relaxation," Linear Algebra and Appl. 2, pp. 199-222 (1969).

[15] G. M. Baudet, "Asynchronous iterative methods for multiprocessors," Journal of the ACM 25, 2, pp. 226-244 (April 1978).

[16] J. P. Fishburn, Analysis of speedup in distributed algorithms, Computer Science Department, University of Wisconsin - Madison (May 1981) Ph.D. thesis.

[17] R. A. Finkel and M. H. Solomon, "The lens interconnection strategy," Proc. 14th Hawaii International Conference on System Sciences, (January 1981).