A Programming Methodology for Designing Block Recursive Algorithms on Various Computer Networks *

Min-Hsuan Fan, Chua-Huang Huang, Yeh-Ching Chung
Department of Information Engineering
Feng Chia University
Taichung, Taiwan, R.O.C.

Abstract

In this paper, we use the tensor product notation as the framework of a programming methodology for designing block recursive algorithms on various computer networks. In our previous works, we propose a programming methodology for designing block recursive algorithms on shared-memory and distributed-memory multiprocessors without considering the interconnection of processors. We extend the work to consider the block recursive algorithms on direct networks and multistage interconnection networks. We use parallel prefix computation as an example to illustrate the methodology. First, we represent the prefix computation problem as a computational matrix which may not be suitable for deriving algorithms on specific computer networks. In this methodology, we add two steps to derive tensor product formulas of parallel prefix algorithms on computer networks: (1) decompose the computational matrix into two submatrices, and (2) construct an augmented matrix. The augmented matrix can be factorized so that each term is a tensor product formula and can fit into a specified network topology. With the augmented matrix, the input data is also extended. It means, in addition to the input data, an auxiliary vector as temporary storage is used. The content of temporary storage is relevant to the decomposition of the original computational matrix. We present the methodology to derive various parallel prefix algorithms on hypercube, omega, and baseline networks and verify correctness of the resulting tensor product formulas using induction.

Keywords: programming methodology, tensor product, block recursive algorithm, parallel processing, parallel prefix, hypercube network, omega network, baseline network.

1 Introduction

Tensor products, also known as Kronecker products [4], have been successfully used to express and implement parallel block recursive algorithms such as fast Fourier transforms [7, 8] and Strassen's matrix multiplication [6, 11]. The tensor product notation is also suitable for expressing data distribution and modeling interconnection networks [9, 10]. The tensor product operations can be mapped to corresponding programming constructs. Therefore, the tensor product notation provides a framework of designing and implementing parallel programs [2, 5].

In our previous works, we propose a programming methodology for designing block recursive algorithms on shared-memory and distributed-memory multiprocessors without considering the interconnection of processors [3]. We extend the work to consider the block recursive algorithms on direct networks and multistage interconnection networks. In this paper, we present a systematic methodology by which tensor product formulas of a given computational problem on specified networks will be derived stepwise. The parallel prefix problem will be used to illustrate the programming methodology. Various parallel prefix algorithms on hypercube, omega, and baseline networks are presented in this paper [14, 15].

The programming methodology is divided into five steps. First, expresses a computational problem in its matrix form. Second, represent the specified network by a network connection matrix. Third, decompose the problem computational matrix into two partial computational matrix and build an augmented problem matrix for the given problem with original input vector and auxiliary input vector. Fourth, try to factorize the augmented matrix of the computational problem into product of tensor product formulas so that each term can fit into the network connection matrix. Finally, we obtain a tensor product formula that represent an algorithm for the parallel prefix problem on the specified network.

In this paper, we use parallel prefix problem as an ex-

* This work was supported in part by National Science Council, Taiwan, R.O.C. under grant NSC90-2213-E-035-024.

ample. The parallel prefix computation is used by many applications such as evaluation of polynomials, solution of linear recurrence equations, carry-look-ahead circuits, radix sorting, quick sorting, and scheduling problems [1].

The organization of this paper is as the following. Section 2 defines the tensor product notation and gives a brief summary of the properties that we will use through this paper. Section 3 presents a programming methodology for a general block recursive algorithm on a specified network and explains the programming methodology using the parallel prefix problem. Section 4 illustrates the generation of parallel prefix algorithms on hypercube network, omega network, and baseline network. The conclusions and future works are given in Section 5.

2 The Tensor Product Notation

In this section, we give a brief overview of the tensor product definition and relevant properties. Tensor product operation is a bilinear form that constructs a block matrix from two matrices.

Definition 2.1 (Tensor product of matrices) *Let $A_{m\times n}$ be an $m \times n$ matrix and $B_{p\times q}$ be a $p \times q$ matrix. The tensor product of A and B is the block matrix obtained by replacing each element $a_{i,j}$ by the matrix $a_{i,j}B_{p\times q}$, i.e., the result is an $mp \times nq$ matrix, defined as*

$$A_{m\times n} \otimes B_{p\times q} = \begin{bmatrix} a_{0,0}B_{p\times q} & \cdots & a_{0,n-1}B_{p\times q} \\ \vdots & \ddots & \vdots \\ a_{n-1,0}B_{p\times q} & \cdots & a_{n-1,n-1}B_{p\times q} \end{bmatrix}_{mp\times nq}.$$

Two important forms of tensor product formulas are when one of the operands is the identity matrix. If the first operand is the identity matrix, i.e., $Y = (I_n \otimes A)X$, it can be interpreted as parallel operations on segments of X and is called the parallel form. If the second operand is the identity matrix, i.e., $Y = (A \otimes I_n)X$, it can be interpreted as vector operations on elements of X and is called the vector form.

Another important operation is a stride permutation.

Definition 2.2 (Stride permutation) $L^{mn}_n(e^m_i \otimes e^n_j) = e^n_j \otimes e^m_i$.

L^{mn}_n is the stride permutation of size mn stride with distance n. When a matrix is stored by rows, the tensor basis $e^m_i \otimes e^n_j$ is isomorphic to $E^{m,n}_{i,j}$. In addition, tensor basis $e^n_j \otimes e^m_i$ is isomorphic to $E^{m,n}_{i,j}$ when a matrix is stored by columns.

The followings are some properties of the tensor products and stride permutations:

1. $A \otimes B \otimes C = (A \otimes B) \otimes C = A \otimes (B \otimes C)$
2. $(A_1 \otimes \cdots \otimes A_k)(B_1 \otimes \cdots \otimes B_k) = (A_1B_1 \otimes \cdots \otimes A_kB_k)$
3. $(A_1 \otimes B_1)(A_2 \otimes B_2) \cdots (A_k \otimes B_k) = (A_1A_2 \cdots A_k \otimes B_1B_2 \cdots B_k)$
4. $\prod_{i=0}^{n-1}(I_n \otimes A_i) = I_n \otimes \prod_{i=0}^{n-1} A_i$
5. $\prod_{i=0}^{n-1}(A_i \otimes I_n) = \prod_{i=0}^{n-1} A_i \otimes I_n$
6. $(L^{mn}_n)^{-1} = L^{mn}_m$, $L^n_n = I_n$
7. $L^{rst}_{rs} = L^{rst}_r L^{rst}_s$
8. $L^{rst}_t = (L^{rt}_t \otimes I_s)(I_r \otimes L^{st}_t)$
9. $L^{rst}_{st} = (I_s \otimes L^{rt}_t)(L^{rs}_s \otimes I_t)$

3 The Programming Methodology

In this section, we propose a programming methodology for deriving block recursive algorithms on various interconnection networks based on the tensor product notation. The programming methodology contains five steps. We explain them as following:

Step 1. Represent the computational problem as a *computational matrix* Q_N. For example, the parallel prefix computation problem can be represented by $Y_N = Q_N X_N$, where X_N is an ordered column vector as the input data, Y_N is the prefix result column vector as the output data, and Q_N is an $N \times N$ computational matrix, where all the lower triangular and diagonal elements of Q_N are 1's, and all the upper triangular elements of Q_N are 0's.

Step 2. Represent the specified interconnection network of N nodes as an $N \times N$ *network connection matrix* H_N. If node i has a link to node j, then the value of matrix element $h_{i,j}$ is one, otherwise the value of $h_{i,j}$ is zero. Suppose P_N is an $N \times N$ matrix and each element of P_N is one or zero. We call a computational matrix P_N fits into a network connection matrix H_N if element $p_{i,j}$ in P_N is one, then $h_{i,j}$ in H_N should also be one, for all i, j from 0 to $N-1$.

For example, if $S = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$, the hypercube network connection matrix is represented as

$$H_{2^n} = \begin{cases} 1, & \text{for } n = 0. \\ I_{2^n} + \sum_{i=0}^{n-1} I_{2^{n-i-1}} \otimes S \otimes I_{2^i}, & \text{for } n > 0. \end{cases}$$

In this network connection matrix, the identity matrix I_{2^n} represents the self-connected links, and $I_{2^{n-i-1}} \otimes S \otimes I_{2^i}$ represents the links along dimension i of the hypercube.

Step 3. Given a network connection matrix as in Step 2, if we use the methodology in our previous paper, it is not possible to factorize the original prefix computational matrix such that each factor fits into the specified network connection matrix. In this step, we try to solve the problem

Proceedings of the International Conference on Parallel Processing Workshops (ICPPW'02)
1530-2016/02 $17.00 © 2002 IEEE

by decomposing the original computational matrix into two partial problem matrices and then augmenting the partial problem matrices so their factorization can fit into the specified network connection matrix. The augmentation is intent to add an auxiliary vector representing temporary storage to the input data. The content of the input vector and the auxiliary vector may affect the decomposition of the computational matrix and the construction of the augmented matrix.

The purpose of the construction of augmented problem matrix is to find a factorization so that each term can fit into the specified network connection matrix. There are two disadvantages of the augmentation. One is more data storage. The other is more communication and computation. The advantage of the augmentation is that we can solve the computation problem in a specified computer network. As a total gain, the result is a more efficient algorithm for the specific interconnection network.

The initial content of the auxiliary vector may simply be zero vector, a copy of the original input vector, or a permutation of the original input vector. The choice of the initial content of the auxiliary vector may affect the construction of the augmented matrix.

An augmented problem matrix of the original prefix computation matrix Q_N can be defined as the following matrix:

$$\begin{bmatrix} Q_N^{11} & Q_N^{12} \\ Q_N^{21} & Q_N^{22} \end{bmatrix},$$

where the size of Q_N^{11}, Q_N^{12}, Q_N^{21}, and Q_N^{22} are $N \times N$. The size of input vector of the augmented problem matrix is $2N$. We will obtain the resulting vector Y_N, in the form of $\begin{bmatrix} Y_N^1 \\ Y_N^2 \end{bmatrix}$, from the augmented computational matrix applying on the original input vector X_N with auxiliary vector Z_N, i.e.,

$$\begin{bmatrix} Y_N^1 \\ Y_N^2 \end{bmatrix} = \begin{bmatrix} Q_N^{11} & Q_N^{12} \\ Q_N^{21} & Q_N^{22} \end{bmatrix} \begin{bmatrix} X_N \\ Z_N \end{bmatrix}.$$

For example, if we initialize the auxiliary vector Z_N to be X_N, we may decompose the original matrix Q_N to Q_N^{11} and Q_N^{12} such that Q_N is equal to $Q_N^{11} + Q_N^{12}$. Therefore, we obtain the equation below:

$$\begin{aligned} & \begin{bmatrix} Q_N^{11} & Q_N^{12} \\ Q_N^{21} & Q_N^{22} \end{bmatrix} \begin{bmatrix} X_N \\ X_N \end{bmatrix} \\ = & \begin{bmatrix} (Q_N^{11} + Q_N^{12})X_N \\ (Q_N^{21} + Q_N^{22})X_N \end{bmatrix} \\ = & \begin{bmatrix} Q_N X_N \\ (Q_N^{21} + Q_N^{22})X_N \end{bmatrix} \\ = & \begin{bmatrix} Y_N \\ (Q_N^{21} + Q_N^{22})X_N \end{bmatrix}. \end{aligned}$$

In this case, $Y_N^1 = Y_N$, which is the computational result, and $Y_N^2 = (Q_N^{21} + Q_N^{22})X_N$, which may be ignored at the end of computation. Therefore, the definition of Q_N^{21} and Q_N^{22} does not affect the computing result. But they may determine whether the augmented matrix can fit into a given network connection. If the specific network is hypercube, we may assign $Q_N^{11} = I_N$ and $Q_N^{12} = Q_N - I_N$.

For another example, if we initialize the auxiliary vector Z_N to be the zero vector, we may decompose the original matrix Q_N to Q_N^{11} and Q_N^{21} such that $Q_N = Q_N^{11} + Q_N^{21}$. We have

$$\begin{bmatrix} Q_N^{11} & Q_N^{12} \\ Q_N^{21} & Q_N^{22} \end{bmatrix} \begin{bmatrix} X_N \\ 0 \end{bmatrix} = \begin{bmatrix} Q_N^{11} X_N \\ Q_N^{21} X_N \end{bmatrix}.$$

We can get the result vector Y_N from the summation of the first half of the output vector and the second half of the output vector, i.e., $Y_N = Q_N^{11} X_N + Q_N^{21} X_N = (Q_N^{11} + Q_N^{21})X_N = Q_N X_N$. The definition of Q_N^{12} and Q_N^{22} does not affect the computing result. But they may determine whether the augmented matrix can fit into a given network connection. If the specific network is hypercube, we may assign $Q_N^{11} = I_N$ and $Q_N^{21} = Q_N - I_N$.

Step 4. In Step 3, we construct an augmented matrix based on the initial value of the input data and the original problem matrix. In this step, we begin to factorize the augmented matrix such that each term should fit into the specified network matrix. In the direct network case, we may use a methodology similar to the methodology in our previous paper.

Considering the hypercube network, we may formulate a matrix equation of $\begin{bmatrix} Q_N^{11} & Q_N^{12} \\ Q_N^{21} & Q_N^{22} \end{bmatrix}$ to factorize the augmented problem matrix into a smaller size of augmented problem matrix with additional pre-operation $\begin{bmatrix} R_N^{11} & R_N^{12} \\ R_N^{21} & R_N^{22} \end{bmatrix}$ and post-operation $\begin{bmatrix} P_N^{11} & P_N^{12} \\ P_N^{21} & P_N^{22} \end{bmatrix}$ as Equation (1):

$$\begin{aligned} & \begin{bmatrix} Q_N^{11} & Q_N^{12} \\ Q_N^{21} & Q_N^{22} \end{bmatrix} = \begin{bmatrix} P_N^{11} & P_N^{12} \\ P_N^{21} & P_N^{22} \end{bmatrix} \\ & \begin{bmatrix} I_{N_1} \otimes Q_{N_2}^{11} \otimes I_{N_3} & I_{N_1} \otimes Q_{N_2}^{12} \otimes I_{N_3} \\ I_{N_1} \otimes Q_{N_2}^{21} \otimes I_{N_3} & I_{N_1} \otimes Q_{N_2}^{22} \otimes I_{N_3} \end{bmatrix} \begin{bmatrix} R_N^{11} & R_N^{12} \\ R_N^{21} & R_N^{22} \end{bmatrix} \end{aligned} \quad (1)$$

where $N = N_1 N_2 N_3$. $\begin{bmatrix} Q_{N_2}^{11} & Q_{N_2}^{12} \\ Q_{N_2}^{21} & Q_{N_2}^{22} \end{bmatrix}$ is the same augmented problem matrix with smaller size $2N_2$. I_{N_1} and I_{N_3} are identity matrices with size N_1 and N_3, respectively. This equation is called the *recursive form* of the augmented problem matrix.

Then, we need to find a solution for the Equation (1) such that P_N^{11}, P_N^{12}, P_N^{21}, P_N^{22}, R_N^{11}, R_N^{12}, R_N^{21}, and R_N^{22} are all fit into the assigned network connection matrix.

For example, if the augmented parallel prefix problem matrix of size $N = 2^n$ with $N_1 = 2$ and $N_3 = 1$. The network is the hypercube network. The augmented parallel

Proceedings of the International Conference on Parallel Processing Workshops (ICPPW'02)
1530-2016/02 $17.00 © 2002 IEEE

prefix problem matrix can be formulated as the following matrix equation:

$$\begin{bmatrix} Q^{11}_{2^n} & Q^{12}_{2^n} \\ Q^{21}_{2^n} & Q^{22}_{2^n} \end{bmatrix} = \begin{bmatrix} P^{11}_{2^n} & P^{12}_{2^n} \\ P^{21}_{2^n} & P^{22}_{2^n} \end{bmatrix} \begin{bmatrix} I_2 \otimes Q^{11}_{2^{n-1}} & I_2 \otimes Q^{12}_{2^{n-1}} \\ I_2 \otimes Q^{21}_{2^{n-1}} & I_2 \otimes Q^{22}_{2^{n-1}} \end{bmatrix} \begin{bmatrix} R^{11}_{2^n} & R^{12}_{2^n} \\ R^{21}_{2^n} & R^{22}_{2^n} \end{bmatrix}$$

Let $Q^{11}_{2^n} = I_{2^n}$, $Q^{12}_{2^n} = Q_{2^n} - I_{2^n}$, $Q^{21}_{2^n} = [0]_{2^n}$, $Q^{11}_{2^n} = [1]_{2^n}$, $R^{11}_{2^n} = I_{2^n}$, $R^{12}_{2^n} = 0$, $R^{21}_{2^n} = 0$, and $R^{22}_{2^n} = I_{2^n}$. There are four matrix equations to be solved as following.

$$I_{2^n} = P^{11}_{2^n}(I_2 \otimes I_{2^{n-1}}) + P^{12}_{2^n}(I_2 \otimes Q^{21}_{2^{n-1}}) \quad (2)$$

$$Q_{2^n} - I_{2^n} = P^{11}_{2^n}(I_2 \otimes (Q_{2^{n-1}} - I_{2^{n-1}})) + P^{12}_{2^n}(I_2 \otimes Q^{22}_{2^{n-1}}) \quad (3)$$

$$Q^{21}_{2^n} = P^{21}_{2^n}(I_2 \otimes I_{2^{n-1}}) + P^{22}_{2^n}(I_2 \otimes Q^{21}_{2^{n-1}}) \quad (4)$$

$$Q^{22}_{2^n} = P^{21}_{2^n}(I_2 \otimes (Q_{2^{n-1}} - I_{2^{n-1}})) + P^{22}_{2^n}(I_2 \otimes Q^{22}_{2^{n-1}}) \quad (5)$$

We solve Equations (2) to (5) and obtain a possible solution $P^{11}_{2^n} = I_{2^n}$, $P^{12}_{2^n} = \begin{bmatrix} 0_{2^{n-1}} & 0_{2^{n-1}} \\ I_{2^{n-1}} & 0_{2^{n-1}} \end{bmatrix}$, $P^{21}_{2^n} = 0$, and $P^{22}_{2^n} = \begin{bmatrix} I_{2^{n-1}} & I_{2^{n-1}} \\ I_{2^{n-1}} & I_{2^{n-1}} \end{bmatrix}$ which all fit into the hypercube network connection matrix. As a result, the solution $P^{ij}_{2^{n-1}}$ and $R^{ij}_{2^{n-1}}$, for $i, j = 1, 2$, all fit into the hypercube network connection matrix. However, Equation (1) is a recursive tensor product formula which is not suitable for program generation.

Step 5. Find the iteration form of the augmented problem matrix. This step is to recursively factorize the augmented problem matrix $\begin{bmatrix} Q^{11}_{2^{n-1}} & Q^{12}_{2^{n-1}} \\ Q^{21}_{2^{n-1}} & Q^{22}_{2^{n-1}} \end{bmatrix}$ with the same factorization in Step 4 until N_2 is small enough such that Q_{N_2} can fit into the network connection matrix. The final factorization is a product of some matrices that each term can fit into the network connection matrix. This equation is called the iteration form of the given computational problem.

Following the example in Step 4, the augmented parallel prefix problem matrix can be expanded as below:

$$\begin{aligned}
&\begin{bmatrix} Q^{11}_{2^n} & Q^{12}_{2^n} \\ Q^{21}_{2^n} & Q^{22}_{2^n} \end{bmatrix} \\
&= \begin{bmatrix} P^{11}_{2^n} & P^{12}_{2^n} \\ P^{21}_{2^n} & P^{22}_{2^n} \end{bmatrix} \begin{bmatrix} I_2 \otimes Q^{11}_{2^{n-1}} & I_2 \otimes Q^{12}_{2^{n-1}} \\ I_2 \otimes Q^{21}_{2^{n-1}} & I_2 \otimes Q^{22}_{2^{n-1}} \end{bmatrix} \\
&= \begin{bmatrix} P^{11}_{2^n} & P^{12}_{2^n} \\ P^{21}_{2^n} & P^{22}_{2^n} \end{bmatrix} \\
&\quad (L^4_2 \otimes I_{2^{n-1}})(I_2 \otimes \begin{bmatrix} Q^{11}_{2^{n-1}} & Q^{12}_{2^{n-1}} \\ Q^{21}_{2^{n-1}} & Q^{22}_{2^{n-1}} \end{bmatrix})(L^4_2 \otimes I_{2^{n-1}}) \\
&= \begin{bmatrix} P^{11}_{2^n} & P^{12}_{2^n} \\ P^{21}_{2^n} & P^{22}_{2^n} \end{bmatrix} \\
&\quad (L^4_2 \otimes I_{2^{n-1}})(I_2 \otimes \begin{bmatrix} P^{11}_{2^{n-1}} & P^{12}_{2^{n-1}} \\ P^{21}_{2^{n-1}} & P^{22}_{2^{n-1}} \end{bmatrix}) \\
&\quad (L^4_2 \otimes I_{2^{n-1}})(L^4_2 \otimes I_{2^{n-1}})(I_2 \otimes L^4_2 \otimes I_{2^{n-2}}) \\
&\quad (I_4 \otimes \begin{bmatrix} Q^{11}_{2^{n-2}} & Q^{12}_{2^{n-2}} \\ Q^{21}_{2^{n-2}} & Q^{22}_{2^{n-2}} \end{bmatrix})(I_2 \otimes L^4_2 \otimes I_{2^{n-2}})(L^4_2 \otimes I_{2^{n-1}}) \\
&= \begin{bmatrix} P^{11}_{2^n} & P^{12}_{2^n} \\ P^{21}_{2^n} & P^{22}_{2^n} \end{bmatrix} \begin{bmatrix} I_2 \otimes P^{11}_{2^{n-1}} & I_2 \otimes P^{12}_{2^{n-1}} \\ I_2 \otimes P^{21}_{2^{n-1}} & I_2 \otimes P^{22}_{2^{n-1}} \end{bmatrix} \\
&\quad (L^4_2 \otimes I_{2^{n-1}})(I_2 \otimes L^4_2 \otimes I_{2^{n-2}}) \\
&\quad (I_4 \otimes \begin{bmatrix} Q^{11}_{2^{n-2}} & Q^{12}_{2^{n-2}} \\ Q^{21}_{2^{n-2}} & Q^{22}_{2^{n-2}} \end{bmatrix})(I_2 \otimes L^4_2 \otimes I_{2^{n-2}})(L^4_2 \otimes I_{2^{n-1}}) \\
&= \cdots \\
&= \prod_{i=1}^{n} \begin{bmatrix} I_{2^{n-i}} \otimes P^{11}_{2^i} & I_{2^{n-i}} \otimes P^{12}_{2^i} \\ I_{2^{n-i}} \otimes P^{21}_{2^i} & I_{2^{n-i}} \otimes P^{22}_{2^i} \end{bmatrix},
\end{aligned}$$

where $P^{11}_{2^i} = I_{2^i}$, $P^{12}_{2^i} = \begin{bmatrix} 0_{2^{i-1}} & 0_{2^{i-1}} \\ I_{2^{i-1}} & 0_{2^{i-1}} \end{bmatrix}$, $P^{21}_{2^i} = [0]_{2^i}$, and $P^{22}_{2^i} = \begin{bmatrix} I_{2^{i-1}} & I_{2^{i-1}} \\ I_{2^{i-1}} & I_{2^{i-1}} \end{bmatrix}$, for $i = 1$ to n.

Suppose each element x_j and z_j of the original input vector X_N and the auxiliary input vector Z_N are allocated in processor j of hypercube network. The final tensor product formula

$$\prod_{i=1}^{n} \begin{bmatrix} I_{2^{n-i}} \otimes P^{11}_{2^i} & I_{2^{n-i}} \otimes P^{12}_{2^i} \\ I_{2^{n-i}} \otimes P^{21}_{2^i} & I_{2^{n-i}} \otimes P^{22}_{2^i} \end{bmatrix}$$

can be interpreted as following. The matrix product notation represents a sequence of the operation steps. Right most matrix will apply to the input data first. The product of n matrices means n sequential steps will be performed. Each step $\begin{bmatrix} I_{2^{n-i}} \otimes P^{11}_{2^i} & I_{2^{n-i}} \otimes P^{12}_{2^i} \\ I_{2^{n-i}} \otimes P^{21}_{2^i} & I_{2^{n-i}} \otimes P^{22}_{2^i} \end{bmatrix}$ means 2^{n-i} copies of $P^{11}_{2^i}$, $P^{12}_{2^i}$, $P^{21}_{2^i}$, and $P^{22}_{2^i}$ are performed in parallel.

Operations $I_{2^{n-i}} \otimes P^{11}_{2^i}$ and $I_{2^{n-i}} \otimes P^{21}_{2^i}$ apply to X_N. Since $P^{11}_{2^i} = I_{2^i}$, operation $I_{2^{n-i}} \otimes P^{11}_{2^i}$ is interpreted as each processor reading the local element of X_N. Since $P^{21}_{2^i} = [0]$, $I_{2^{n-i}} \otimes P^{21}_{2^i}$ is interpreted as no-operation on the local element of X_N.

Operations $I_{2^{n-i}} \otimes P^{12}_{2^i}$ and $I_{2^{n-i}} \otimes P^{22}_{2^i}$ apply to Z_N. Since $P^{12}_{2^i} = \begin{bmatrix} 0_{2^{i-1}} & 0_{2^{i-1}} \\ I_{2^{i-1}} & 0_{2^{i-1}} \end{bmatrix}$, operation $I_{2^{n-i}} \otimes P^{12}_{2^i}$ is interpreted as partitioning processors into 2^{n-i} groups. In each group, the processors in the first half of the group move the local data in Z_N along dimension $i-1$ of the hypercube network to the processors in the second half of the group, and then the receiving processors adding the received data to the local data in X_N.

Proceedings of the International Conference on Parallel Processing Workshops (ICPPW'02)
1530-2016/02 $17.00 © 2002 IEEE

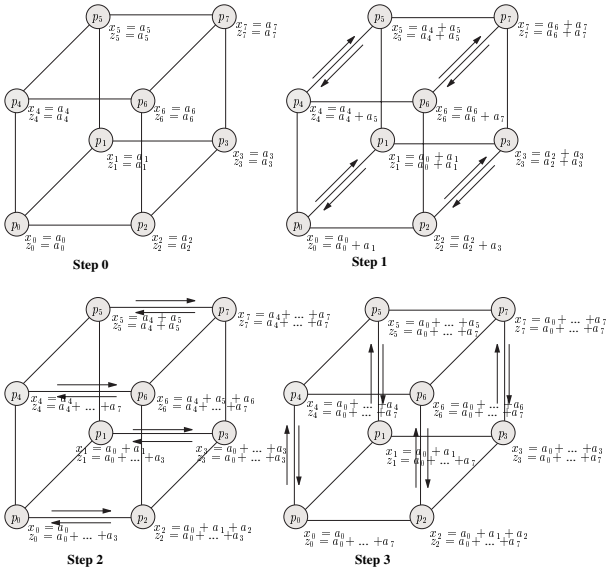

Figure 1. Parallel prefix algorithm on hypercube for $N = 8$

Since $P_{2^i}^{22} = \begin{bmatrix} I_{2^{i-1}} & I_{2^{i-1}} \\ I_{2^{i-1}} & I_{2^{i-1}} \end{bmatrix}$, operation $I_{2^{n-i}} \otimes P_{2^i}^{22}$ is interpreted as partitioning processors into 2^{n-i} groups. In each group, the processors in the first half group and the second half group exchange the local data in Z_N along dimension $i-1$ of the hypercube network, and then adding the received data to the local data in Z_N.

We demonstrate this step by an example of size 8 in Figure 1. The SPMD program of this parallel prefix algorithm is given in Figure 2. The SPMD program is suitable for fine-grained computation. Each processor has only two local variables x and z, an output variable y, and a temporary variable $temp$. As shown in Figure 1, the final result of Z_i in each processor is the summation of the input data.

From the above five steps, we have derived an algorithm for a computation problem on a specified network. The algorithm can be represented by a tensor product formula and translated to program statements directly.

4 More Parallel Prefix Algorithms by The Methodology

In this section, we use the methodology in the previous section to derive different algorithms for parallel prefix problem on the hypercube network, the omega network, and the baseline network.

```
z = x
for i = 1 to n
 offset = my_rank - (my_rank mod 2^i)
 if (my_rank - offset < 2^(i-1) ) then
  send(processor[my_rank + 2^(i-1)], z)
  receive(processor[my_rank + 2^(i-1)], temp)
  z = z + temp
 end if
 if (my_rank - offset >= 2^(i-1)) then
  send(processor[my_rank - 2^(i-1)], z)
  receive(processor[my_rank - 2^(i-1)], temp)
  x = x + temp
  z = z + temp
 end if
end for
y = x
```

Figure 2. SPMD program for the parallel prefix algorithm on hypercube

4.1 Another Parallel Prefix Algorithm on Hypercube Network

Based on the methodology in Section 3, the auxiliary vector Z and the matrices $Q_{2^n}^{11}$, $Q_{2^n}^{21}$, $Q_{2^n}^{12}$, and $Q_{2^n}^{22}$ can be defined in many ways. Alternative definition may result in different algorithms.

Let the auxiliary vector Z_{2^n} be 0 and the augmented problem matrix $\begin{bmatrix} Q_{2^n}^{11} & Q_{2^n}^{12} \\ Q_{2^n}^{21} & Q_{2^n}^{22} \end{bmatrix}$ satisfy the condition $Q_{2^n}^{11} + Q_{2^n}^{21} = Q_{2^n}$, where Q_{2^n} is the prefix computation matrix. Let $Q_{2^n}^{11} = I_{2^n}$, $Q_{2^n}^{21} = Q_{2^n} - I_{2^n}$, $Q_{2^n}^{12} = [0]_{2^n}$, and $Q_{2^n}^{22} = [1]_{2^n}$. We also set $P_{2^n}^{11} = I_{2^n}$, $P_{2^n}^{12} = 0$, $P_{2^n}^{21} = 0$, and $P_{2^n}^{22} = I_{2^n}$ which all fit into the hypercube network connection matrix.

The recursive form of the augmented parallel prefix problem matrix can be formulated as the following matrix equation:

$$\begin{bmatrix} I_{2^n} & [0]_{2^n} \\ Q_{2^n} - I_{2^n} & [1]_{2^n} \end{bmatrix} = \begin{bmatrix} I_2 \otimes I_{2^{n-1}} & I_2 \otimes [0]_{2^{n-1}} \\ I_2 \otimes (Q_{2^{n-1}} - I_{2^{n-1}}) & I_2 \otimes [1]_{2^{n-1}} \end{bmatrix} \begin{bmatrix} R_{2^n}^{11} & R_{2^n}^{12} \\ R_{2^n}^{21} & R_{2^n}^{22} \end{bmatrix}.$$

There are four matrix equations to be solved as following:

$$I_{2^n} = (I_2 \otimes I_{2^{n-1}}) R_{2^n}^{11} \tag{6}$$

$$[0]_{2^n} = (I_2 \otimes I_{2^{n-1}}) R_{2^n}^{12} \tag{7}$$

$$Q_{2^n} - I_{2^n} = (I_2 \otimes (Q_{2^{n-1}} - I_{2^{n-1}})) R_{2^n}^{11} + (I_2 \otimes [1]_{2^{n-1}}) R_{2^n}^{21} \tag{8}$$

$$[1]_{2^n} = (I_2 \otimes (Q_{2^{n-1}} - I_{2^{n-1}})) R_{2^n}^{12} + (I_2 \otimes [1]_{2^{n-1}}) R_{2^n}^{22} \tag{9}$$

Proceedings of the International Conference on Parallel Processing Workshops (ICPPW'02)
1530-2016/02 $17.00 © 2002 IEEE

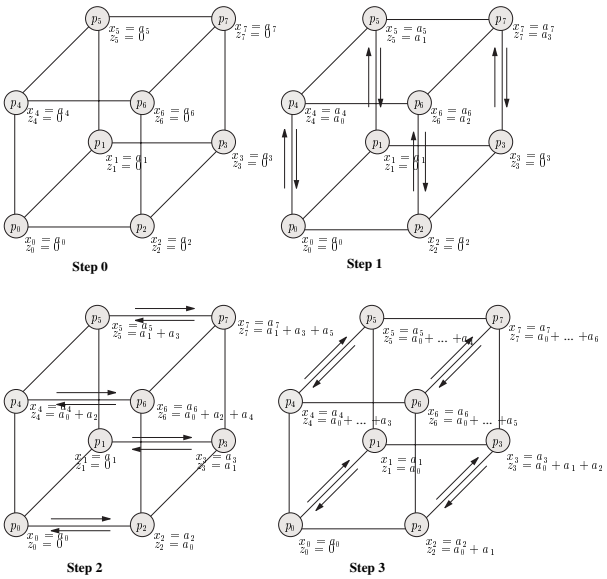

Figure 3. Parallel prefix algorithm on hypercube for $N = 8$

Solving Equation (6) to (9), we obtain one feasible solution, $R_{2^n}^{11} = I_{2^n}$, $R_{2^n}^{12} = 0$, $R_{2^n}^{21} = \begin{bmatrix} 0_{2^{n-1}} & 0_{2^{n-1}} \\ I_{2^{n-1}} & 0_{2^{n-1}} \end{bmatrix}$, and $R_{2^n}^{22} = \begin{bmatrix} I_{2^{n-1}} & I_{2^{n-1}} \\ I_{2^{n-1}} & I_{2^{n-1}} \end{bmatrix}$, which all fit into the hypercube network connection matrix.

The iteration form of the augmented prefix computation problem is as following:

$$\begin{bmatrix} Q_{2^n}^{11} & Q_{2^n}^{12} \\ Q_{2^n}^{21} & Q_{2^n}^{22} \end{bmatrix} = \prod_{i=n}^{1} \begin{bmatrix} I_{2^{n-i}} \otimes R_{2^i}^{11} & I_{2^{n-i}} \otimes R_{2^i}^{12} \\ I_{2^{n-i}} \otimes R_{2^i}^{21} & I_{2^{n-i}} \otimes R_{2^i}^{22} \end{bmatrix},$$

where $R_{2^i}^{11} = I_{2^i}$, $R_{2^i}^{12} = 0$, $R_{2^i}^{21} = \begin{bmatrix} 0_{2^{i-1}} & 0_{2^{i-1}} \\ I_{2^{i-1}} & 0_{2^{i-1}} \end{bmatrix}$, and $R_{2^i}^{22} = \begin{bmatrix} I_{2^{i-1}} & I_{2^{i-1}} \\ I_{2^{i-1}} & I_{2^{i-1}} \end{bmatrix}$.

We demonstrate this step by an example of size 8 in Figure 3.

4.2 Parallel Prefix Algorithm on Omega Network

In a multistage interconnection network, the network connection matrix between each stage is a permutation matrix. For a multistage interconnection network of $N = 2^n$ input lines, each stage contains 2^{n-1} switching elements. We use $S_{2^n} = I_{2^{n-1}} \otimes s_2$ to represent a switching stage and operation $s_2 = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$ to represent a switching element, where $a_{ij} = 0$ or 1, for $1 \le i, j \le 2$, depending on the computation performed in the switching element. A permutation between two switching stages is required to connect the output lines of a switching stage to the input lines of its adjacent stage. In addition, a pre-permutation and a post-permutation may be required to rearrange the input data and output data, respectively. For example, the pre-permutation and the permutation between switching stages of the omega network are the stride permutation $L_{2^{n-1}}^{2^n}$. The omega network can be represented in the following tensor product formula:

$$\Omega_N = \prod_{i=0}^{n-1} (I_{2^{n-1}} \otimes s_2) L_{2^{n-1}}^{2^n}.$$

For the parallel prefix computation on the omega network, we build an augmented problem matrix $\begin{bmatrix} Q_{2^n}^{11} & Q_{2^n}^{12} \\ Q_{2^n}^{21} & Q_{2^n}^{22} \end{bmatrix}$ and let the auxiliary input vector Z_{2^n} be 0, then $\begin{bmatrix} Q_{2^n}^{11} & Q_{2^n}^{12} \\ Q_{2^n}^{21} & Q_{2^n}^{22} \end{bmatrix} \begin{bmatrix} X_{2^n} \\ 0 \end{bmatrix} = \begin{bmatrix} Q_{2^n}^{11} X_{2^n} \\ Q_{2^n}^{21} X_{2^n} \end{bmatrix}$. If $Q_{2^n}^{11} + Q_{2^n}^{21} = Q_{2^n}$, we may get the result vector Y_{2^n} from the summation of the first half of the output vector and the second half of the output vector, i.e., $Y_{2^n} = Q_{2^n}^{11} X_{2^n} + Q_{2^n}^{21} X_{2^n}$. We set $Q_{2^n}^{11} = I_{2^n}$ and $Q_{2^n}^{21} = Q_{2^n} - I_{2^n}$.

To design the parallel prefix algorithm on the omega network, we will rewrite the augmented matrix to the following form:

$$\begin{bmatrix} Q_{2^n}^{11} & Q_{2^n}^{12} \\ Q_{2^n}^{21} & Q_{2^n}^{22} \end{bmatrix} = \prod_{i=0}^{n-1} \begin{bmatrix} (I_{2^{n-1}} \otimes A_2) L_{2^{n-1}}^{2^n} & (I_{2^{n-1}} \otimes B_2) L_{2^{n-1}}^{2^n} \\ (I_{2^{n-1}} \otimes C_2) L_{2^{n-1}}^{2^n} & (I_{2^{n-1}} \otimes D_2) L_{2^{n-1}}^{2^n} \end{bmatrix} \quad (10)$$

where A_2, B_2, C_2, and D_2 are computations to be performed in the switching elements. We need to derive A_2, B_2, C_2, and D_2. The definition of $Q_{2^n}^{12}$ and $Q_{2^n}^{22}$ does not affect the computation result. Their definition is determined by the values of A_2, B_2, C_2, and D_2.

Considering the omega network of size 2, i.e., $n = 1$, we rewrite the augmented matrix as following:

$$\begin{bmatrix} Q_2^{11} & Q_2^{12} \\ Q_2^{21} & Q_2^{22} \end{bmatrix} = \begin{bmatrix} A_2 & B_2 \\ C_2 & D_2 \end{bmatrix} \quad (11)$$

Since $Q_2 = Q_2^{11} + Q_2^{21}$, we set $A_2 = Q_2^{11} = I_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ and obtain $C_2 = Q_2^{21} = Q_2 - Q_2^{11} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}$.

Next, considering the omega network of size 4, we again

Proceedings of the International Conference on Parallel Processing Workshops (ICPPW'02)
1530-2016/02 $17.00 © 2002 IEEE

rewrite the augmented matrix as following:

$$= \begin{bmatrix} Q_4^{11} & Q_4^{12} \\ Q_4^{21} & Q_4^{22} \end{bmatrix} \begin{bmatrix} (I_2 \otimes A_2)L_2^4 & (I_2 \otimes B_2)L_2^4 \\ (I_2 \otimes C_2)L_2^4 & (I_2 \otimes D_2)L_2^4 \end{bmatrix} \begin{bmatrix} (I_2 \otimes A_2)L_2^4 & (I_2 \otimes B_2)L_2^4 \\ (I_2 \otimes C_2)L_2^4 & (I_2 \otimes D_2)L_2^4 \end{bmatrix} \quad (12)$$

From Equation (12), we have

$$\begin{aligned} Q_4^{11} &= (I_2 \otimes A_2)L_2^4(I_2 \otimes A_2)L_2^4 + (I_2 \otimes B_2)L_2^4(I_2 \otimes C_2)L_2^4 \\ &= I_4 + (I_2 \otimes B_2)(C_2 \otimes I_2) \\ &= I_4 + (C_2 \otimes B_2). \end{aligned}$$

Since $Q_4^{11} = I_4$, we obtain a trivial solution of $B_2 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$. From Equation (12), we also have

$$\begin{aligned} Q_4^{21} &= (I_2 \otimes C_2)L_2^4(I_2 \otimes A_2)L_2^4 + (I_2 \otimes D_2)L_2^4(I_2 \otimes C_2)L_2^4 \\ &= (I_2 \otimes C_2) + (I_2 \otimes D_2)(C_2 \otimes I_2) \\ &= (I_2 \otimes C_2) + (C_2 \otimes D_2) \end{aligned}$$

Since $Q_4^{21} = Q_4 - I_4$, we obtain $D_2 = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$. Furthermore, from the solution of A_2, B_2, C_2, and D_2, we can obtain $Q_{2^n}^{12} = [0]_{2^n}$, and $Q_{2^n}^{22} = [1]_{2^n}$. The correctness of Equation (10) can be verified by induction.

We demonstrate this algorithm by an example of size 8 in Figure 4.

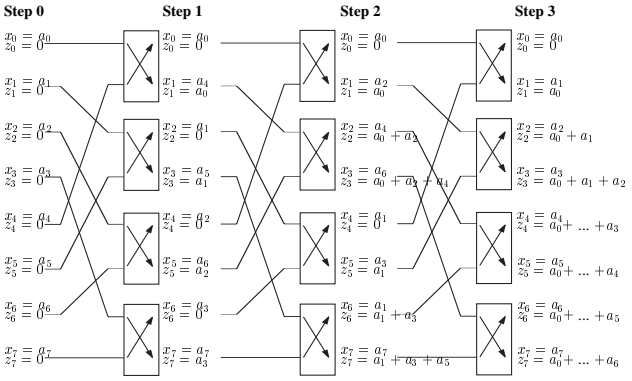

Figure 4. Parallel prefix algorithm on omega network for $N = 8$

4.3 Parallel Prefix Algorithm on Baseline Network

The baseline network is defined as the following tensor product formula:

$$B_N = \prod_{i=0}^{n-1} (I_{2^i} \otimes L_2^{2^{n-i}})(I_{2^{n-1}} \otimes s_2).$$

We choose the auxiliary vector Z_{2^n} to be 0. The augmented matrix $\begin{bmatrix} Q_{2^n}^{11} & Q_{2^n}^{12} \\ Q_{2^n}^{21} & Q_{2^n}^{22} \end{bmatrix}$ must satisfy the condition $Q_{2^n}^{11} + Q_{2^n}^{21} = Q_{2^n}$, where Q_{2^n} is the prefix computation matrix. We omit the detailed steps and show the factorization of the augmented matrix as the following:

$$\begin{bmatrix} Q_{2^n}^{11} & Q_{2^n}^{12} \\ Q_{2^n}^{21} & Q_{2^n}^{22} \end{bmatrix} = \prod_{i=0}^{n-1} \left(\begin{bmatrix} I_{2^n} & 0 \\ 0 & I_{2^i} \otimes L_2^{2^{n-i}} \end{bmatrix} \begin{bmatrix} A_{2^n} & B_{2^n} \\ C_{2^n}^i & D_{2^n} \end{bmatrix} \right) \quad (13)$$

where

$$\begin{aligned} A_{2^n} &= I_{2^{n-1}} \otimes I_2 \\ B_{2^n} &= I_{2^{n-1}} \otimes 0_2 \\ C_{2^n}^i &= \begin{cases} I_{2^i} \otimes \left(\left(I_{2^{n-i-2}} \otimes \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix} \right) \oplus \left(I_{2^{n-i-2}} \otimes \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \right) \right) & \text{for } i = 0 \text{ to } n-2 \\ I_{2^{n-1}} \otimes \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} & \text{for } i = n-1 \end{cases} \\ D_{2^n} &= I_{2^{n-1}} \otimes \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \end{aligned}$$

The correctness of Equation (13) can be verified using induction. The final tensor product formula (13) represents a parallel prefix algorithm implemented on a baseline network. If we expand the product of Equation (13), the original input vector is applied on I_{2^n} in each stage, i.e., the input vector remains unchanged. In the baseline network, this fact can be implemented as each switching element containing two local registers and storing the input data X_N statically.

The auxiliary vector Z_N is manipulated according to the application of $C_{2^n}^i$ on X_N and D_{2^n} on Z_N and its elements are transmitted according to the stride permutation of the baseline network. An example of the this algorithm for $N = 8$ is demonstrated in Figure 5.

5 Conclusions

In this paper, we present a programming methodology for designing block recursive algorithms on various computer networks. We employ the tensor product notation to formulate computational problems and derive different algorithms on various computer networks.

The key idea of the programming methodology is to augment the problem computation matrix and accompanied with an auxiliary vector. Then factorize the augmented matrix such that each term should fit into a specified network connection matrix. The factorization steps require solving of matrix equations. Since there are more matrix variables than the number of matrix equations, the solutions of the matrix variables are not unique. A key issue of deriving

Proceedings of the International Conference on Parallel Processing Workshops (ICPPW'02)
1530-2016/02 $17.00 © 2002 IEEE

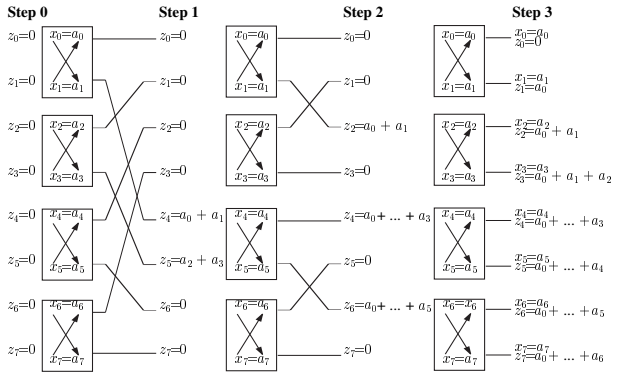

Figure 5. Parallel prefix algorithm on baseline network for $N = 8$

proper algorithms for a specific network is to find solutions that fit into interconnection of the network.

The programming methodology does not only provide a formal approach for designing parallel algorithms on specific interconnection networks, both direct and multistage networks, but it can also be used for designing VLSI circuits. In the future, we will extend the methodology to generate VHDL programs and thus to design circuits for special purpose applications such as digital signal processing and image processing [12, 13].

References

[1] G. E. Blelloch. Prefix sums and their applications. Technical Report CMU-CS-90-190, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, November 1990.

[2] D. L. Dai, S. K. S. Gupta, S. D. Kaushik, J. H. Lu, R. V. Singh, C. H. Huang, P. Sadayappan, and R. W. Johnson. Extent: A portable programming environment for designing and implementing high-performance block-recursive algorithms. In *Proceedings of Supercomputing '94*, pages 49–58, Los Alamitos, USA, 1994. IEEE Comput. Soc. Press.

[3] M.-H. Fan, C.-H. Huang, Y.-C. Chung, J.-S. Liu, and J.-Z. Lee. A programming methodology for designing parallel prefix algorithms. In *Proceedings of the 2001 International Conference on Parallel Processing*, pages 463–470, Los Alamitos, USA, 2001. IEEE Comput. Soc. Press.

[4] A. Graham. *Kronecker Products and Matrix Calculus: With Applications*. Ellis Horwood Limited, 1981.

[5] S. K. S. Gupta, C.-H. Huang, P. Sadayappan, and R. W. Johnson. A framework for generating distributed-memory parallel programs for block recursive algorithms. *J. Parallel and Distributed Computing*, 34:137–153, 1996.

[6] C.-H. Huang, J. R. Johnson, and R. W. Johnson. A tensor product formulation of Strassen's matrix multiplication algorithm. *Appl. Math Letters*, 3(3):104–108, 1990.

[7] J. R. Johnson, R. W. Johnson, D. Rodriguez, and R. Tolimieri. A methodology for designing, modifying and implementing Fourier transform algorithms on various architectures. *Circuits Systems Signal Process*, 9(4):450–500, 1990.

[8] R. W. Johnson, C.-H. Huang, and J. R. Johnson. Multilinear algebra and parallel programming. *The Journal of Supercomputing*, 5(2–3):189–217, October 1991.

[9] S. D. Kaushik, S. Sharma, and C.-H. Huang. An algebraic theory for modeling multistage interconnection networks. *Journal of Information Science and Engineering*, 9(1):1–26, 1993.

[10] S. D. Kaushik, S. Sharma, C.-H. Huang, J. R. Johnson, R. W. Johnson, and P. Sadayappan. An algebraic theory for modeling direct interconnection networks. *Journal of Information Science and Engineering*, 12(1):25–49, 1996.

[11] B. Kumar, C.-H. Huang, P. Sadayappan, and R. W. Johnson. A tensor product formulation of Strassen's matrix multiplication algorithm with memory reduction. *Scientific Programming*, 4(4):275–289, Winter 1995.

[12] S. Mazor and P. Langstraat. *A Guide to VHDL*. Kluwer Academic Publishers, 1992.

[13] G. X. Ritter and P. D. Gader. Image algebra techniques and parallel image processing. *J. Parallel and Distributed Computing*, 4:7–44, 1987.

[14] Y. Saad and M. Schultz. Data communication yypercubes. *Journal of Parallel and Distributed Computing*, 6:115–135, 1989.

[15] C.-L. Wu and T. Feng. On a class of multistage interconnection networks. *IEEE Transactions on Computers*, 29(8):801–810, 1980.

Proceedings of the International Conference on Parallel Processing Workshops (ICPPW'02)
1530-2016/02 $17.00 © 2002 **IEEE**