An Optimal Multiplication Algorithm on Reconfigurable Mesh

Ju-wook Jang, Heonchul Park, and Viktor K. Prasanna, *Fellow, IEEE*

Abstract—An $O(1)$ time algorithm to multiply two N-bit binary numbers using an $N \times N$ bit-model of reconfigurable mesh is shown. It uses optimal mesh size and it improves previously known results for multiplication on the reconfigurable mesh. The result is obtained by using novel techniques for data representation and data movement and using multidimensional Rader Transform. The algorithm is extended to result in AT^2 optimality over $1 \le T \le \sqrt{N}$ in a variant of the bit-model of VLSI.

Index Terms—Integer multiplication, reconfigurable mesh, optimal algorithm, area-time trade off, VLSI architecture.

1 Introduction

The *reconfigurable mesh* is a two-dimensional mesh of processors connected by *reconfigurable buses* [17]. Though the buses outside the Processing Elements (PEs) are fixed, the internal connection between the I/O ports of each PE can be reconfigured by individual PEs during the execution of algorithms.

The reconfigurable mesh captures salient features from a variety of sources, including the Content Addressable Array Parallel Processor (CAAPP), the Configurable Highly Parallel Computer (CHiP), the polymorphic-torus network, and the bus automaton. It consists of an array of processors interconnected by a reconfigurable bus system, which can be used to dynamically obtain various interconnection patterns between the processors. Several reconfigurable meshes are being built. The Image Understanding Architecture (IUA) is a multilevel system designed for supporting research in real-time image understanding and knowledge-based computer vision [32], [33], [34]. The lowest level of the IUA is the CAAPP, a 512×512 square grid of bit serial processors intended to perform low-level image processing. Each CAAPP processor is connected to its four nearest neighbors, as in a standard mesh. In addition, the CAAPP provides a *coterie network* for communication, which consists of a grid-shaped bus and a set of locally controllable switches. The DARPA Integrated Image Understanding Benchmark has been performed on the IUA [35]. A 1/64th prototype of the system has been built at the Hughes Research Labs. Another reconfigurable mesh having 512 PEs has been built by NEC [13]. This architecture consists of an array of processors, and a message passing network. Each processor consists of a DSP chip. The message passing network is an array of message routing chips. The message routing chip, called the Gate Chip, provides connections between the four I/O ports and the DSP chip at the grid.

Parallel algorithms have been developed on the reconfigurable mesh for graph problems [17], [18], [29], for image processing [11], [19], [18], for geometric problems [25], for arithmetic problems [3], [31], and for sorting [3], [8], [15], [20], [25], [30].

In this paper, we show that multiplication of two N bit numbers can be performed in $O(1)$ time on an $N \times N$ bit-model of *reconfigurable mesh.* Previously known result for constant time multiplication on the reconfigurable mesh uses $N^2 \times N^2$ PEs [31]. We also show that our result can be extended to provide area-time tradeoffs to satisfy AT^2 optimality over $1 \le T \le \sqrt{N}$ in a variant of the bit-model of VLSI.

The rest of this paper is organized as follows. In Section 2, a brief description of the architecture is given along with illustration of some basic operations. Section 3.1 discusses Rader Transform. The multiplication algorithm is described in detail in Section 3.2. Section 3.3 shows area-time tradeoffs and Section 3.4 discusses multiplication on related reconfigurable mesh models. Section 4 concludes this paper.

2 Architecture and Basic Operations

The word-model of the reconfigurable mesh has been defined earlier [17]. In this paper, we employ the bit-model.

2.1 Bit-Model of Reconfigurable Mesh

A 6×6 reconfigurable mesh is shown in Fig. 1. $PE(i, j)$ denotes the processing element at the intersection of the ith row and the jth column. Each PE has $O(1)$ words of storage. Each word consists of $O(1)$ bits. The Processing Elements (PEs) can perform basic arithmetic and logic operations on $O(1)$ bits of data in unit time. Each PE has four I/O ports (E, W, N, S). The internal connection between the four ports of a PE can be configured to form subbuses of various shapes during the execution of of an algorithm. A subbus is a collection of pieces of bus which are connected together. Fig. 2 shows some possible connection patterns. For example, {SW, NE} represents the configuration in which S(South) port

- J-w. Jang is with the Department of Electrical Engineering, Sogang University, Seoul, Korea 121-742.
- H. Park is with Samsung Electronics Co. Ltd., Seoul, Korea.
- V.K. Prasanna is with the Department of EE-Systems, EEB 200C, University of Southern California, Los Angeles, CA 90089-2562. E-mail: prasanna@ganges.usc.edu. http://www.usc.edu/dept/ceng/prasanna/home.html.

Manuscript received 28 Apr. 1992.
For information on obtaining reprints of this article, please send e-mail to: transpds@computer.org, and reference IEEECS Log Number D95269.

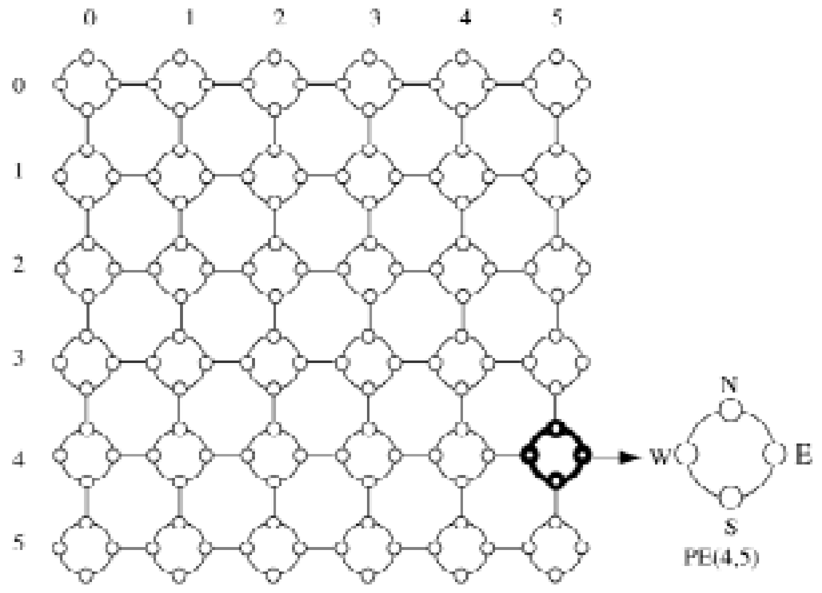

Fig. 1. A 6 × 6 reconfigurable mesh.

is connected to W(West) port while N(North) port is connected to E(East) port. Through any I/O port, each PE can broadcast to or read from the subbus connecting the I/O port. At most, one processor connected to a subbus is allowed to broadcast to the subbus to which it is connected at any given time. Each subbus can carry one of two signals: a *1-signal* or a *0-signal*. The presence of a *1-signal* at a port is represented by a dark circle. Unit time is assumed for broadcasting on the buses, for performing an arithmetic or logic operation on $O(1)$ bits of data or for reconfiguring the connections among the ports in the PEs. Throughout this paper, reconfigurable mesh is used to denote the above bit-model.

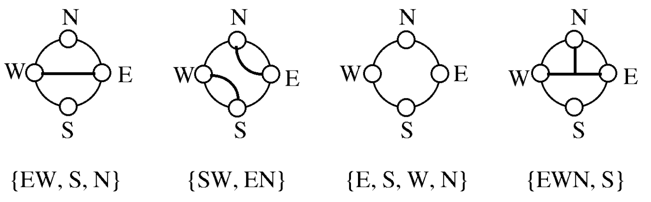

Fig. 2. Some patterns of internal connection among the I/O ports of a PE.

2.2 Number Representations

Input data as well as intermediate results can be represented using the I/O ports of a group of PEs (see Fig. 3 for three representations of number 3):

- BIN Representation: Uses log N PEs to represent a number in $[0, N - 1]$ with a particular (N, S, E, or W) port of the kth PE having a *1-signal* iff the kth bit of the log N bit representation of number i is equal to 1, $0 \leq k \leq \log N - 1$.[1]
- 1UN Representation: Uses N PEs to represent a number i in $[0, N - 1]$ with a particular port of the first $i + 1$ PEs having a *1-signal* and the corresponding port of the rest of the PEs having a *0-signal*.
- 2UN Representation: Uses N PEs to represent a number i in $[0, N]$ with a particular port of a set of i PEs having a *1-signal* and the corresponding port of the rest of the PEs having a *0-signal*. Note that the 2UN representation of a number is not unique. Usually, the 2UN representation is a given 0/1 sequence from which the number of 1s is to be counted during the execution of the algorithm.

2.3 Some Operations on the Reconfigurable Mesh

A number of operations can be performed quickly on the reconfigurable mesh. Broadcast of data along the rows or columns can be performed in $O(1)$ time. A permutation on

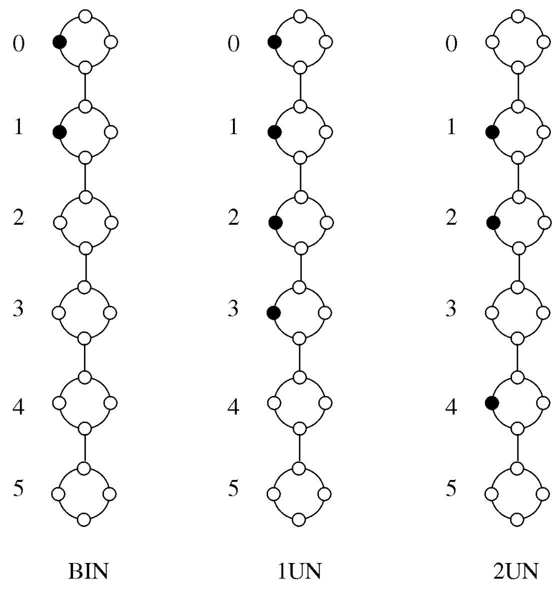

Fig. 3. Three representations of number 3.

1. All logarithms in this paper are to base 2.

N data in any row or column of an $N \times N$ reconfigurable mesh can be performed in $O(1)$ time. Given an integer in $[0, N - 1]$ in the 2UN representation, it can be converted into its 1UN representation in $O(1)$ time using an $N \times N$ bit-model of reconfigurable mesh. For completeness, the conversion technique is described in the following:

LEMMA 1. *Given a 2UN representation of an integer in* $[0, N + 1]$ *in a row, it can be converted to its 1UN representation in* $O(1)$ *time using a* $N \times N$ *reconfigurable mesh.*

Step 1: Let $PE(i, j)$, $two(k)$ and $one(l)$ represent the (i, j)th PE, the kth bit of a given 2UN input and the lth bit of its 1UN output, respectively, where $0 \le i, j, k, l \le N - 1$. Initially, $two(k)$ is given in $PE(0, k)$, $0 \le k \le N - 1$.
Step 2: $PE(0, k)$ broadcasts $two(k)$ along column k, $0 \le k \le N - 1$.
Step 3: All PEs receiving a *1-signal* in Step 2 set their configuration to $\{SW, EN\}$ and the PEs receiving a *0-signal* set their configuration to $\{EW, N, S\}$
Step 4: PE(0, 0) broadcasts a *1-signal* to its W port. Also, if $PE(0, k)$ has $two(k) = 1$, then it broadcasts a *1-signal* to its N port.
Step 5: Check the E port of $PE(u, N - 1)$, $0 \le u \le N - 1$ for 1 – *signal*. The output *one* (l) is represented by the E port of the $PE(l, N - 1)$, $0 \le l \le N - 1$. For example, if $two(k)$, $0 \le k \le N - 1$ is 01011 then the E port of $PE(u, N - 1)$, $0 \le u \le 3$ will have a *1-signal*, and the rest of the ports will have a *0-signal*. This is the 1UN representation of 3.

Addition of two k-bit binary numbers can be performed in $O(1)$ time using a $1 \times k$ reconfigurable mesh as follows (see Fig. 4). Assume $PE(0, i)$ has a_i, b_i, $0 \le i \le k - 1$. $PE(0, i)$ does the following:

- if $(a_i, b_i) = (1, 1)$ set configuration to {E, W, N, S} and broadcast a *1-signal* on its E port.
- if $(a_i, b_i) = (0, 0)$ set configuration to {E, W, N, S} and broadcast a *0-signal* on its E port.
- if $(a_i, b_i) = (1, 0)$ or $(0, 1)$ set configuration to {EW, N, S}.

Now, apply a *0-signal* at the W port of $PE(0, 0)$. $PE(0, i)$, $0 \le i \le k - 1$, sets $c_i = 1$ if a *1-signal* is received at its W port. $PE(0, i)$ computes output $z_i = a_i \oplus b_i \oplus c_i$, $0 \le i \le k - 1$.

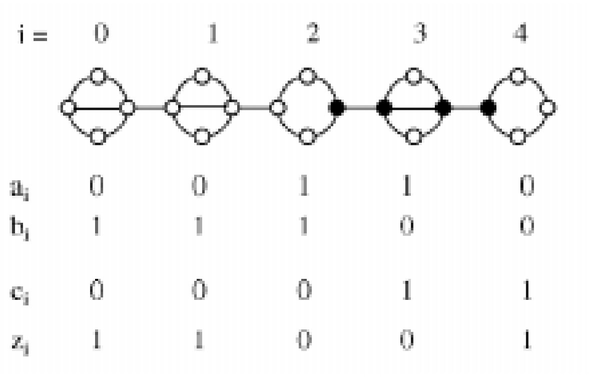

Fig. 4. Addition of two k-bit numbers for $k = 5$.

Given N k-bit binary numbers, $1 \le k \le N$, the *addition problem* is to add these numbers into a $k + \log N$ bit binary number. Wang, Chen and Li [31] solve this problem in $O(1)$ time on an $Nk \times Nk$ reconfigurable mesh. Ben-Asher, Peleg, Ramaswami and Schuster [3] show an algorithm which adds N N-bit numbers in $O(\log^* N)$ time on an $N \times N \times N$ (three-dimensional) reconfigurable mesh. Here, for the sake of completeness, we show an $O(1)$ time solution to the *addition problem* on $N \times Nk$ reconfigurable mesh.

LEMMA 2. *Given N k-bit binary numbers in the BIN representation,* $1 \le k \le N$, *these numbers can be added in* $O(1)$ *time on an* $N \times Nk$ *reconfigurable mesh.*

PROOF. Without loss of generality, assume N is a power of 2. Let the N k-bit numbers be X_i, $0 \le i \le N - 1$. These numbers are represented as follows:

$$X_i = x_{i,0} + x_{i,1}2^1 + x_{i,2}2^2 + \ldots + x_{i,k-1}2^{k-1},$$
$$x_{i,j} \in \{0, 1\}, 0 \le j \le k - 1.$$

Then, the sum of these numbers can be represented using $k + \log N$ bits as:

$$\sum_{i=0}^{N-1} X_i = z_0 + z_1 2^1 + z_2 2^2 + \ldots + z_{k-1+\log N} 2^{k-1+\log N},$$
$$z_j \in \{0,1\}, 0 \le j \le k - 1 + \log N.$$

The output z_j, $0 \le j \le k - 1 + \log N$, can be computed as follows:

$$\begin{aligned} \text{Let}\quad S_j &= \sum_{l=0}^{N-1} x_{l,j}, && 0 \le j \le k-1, \\ S_j &= 0, && k \le j \le k-1+\log N, \\ C_{j+1} &= \left(S_j + C_j\right) div\ 2, && 0 \le j \le k-2+\log N, \text{and} \\ C_0 &= 0 \end{aligned}$$

$$\text{Then, } z_j = \left(S_j + C_j\right) mod\ 2,\ 0 \le j \le k - 1 + \log N.$$

The *mod* 2 and *div* 2 functions give the remainder and the quotient respectively, when the input number is divided by two. If we can compute the carries, C_j, for $1 \le j \le k - 1 + \log N$, in $O(1)$ time, then the $k + \log N$ bit representation of the sum, z_j, $0 \le j \le k - 1 + \log N$, can be obtained in $O(1)$ time. Initially, input $x_{i,j}$ is given in $PE(i, j2N)$ as shown in Fig. 5. At the end of the computation, the output z_j will be available in $PE(0, j)$, $0 \le j \le k - 1 + \log N$.

For ease of explanation we use a $2N \times 2Nk$ reconfigurable mesh. The mesh is partitioned into k blocks of size $2N \times 2N$. Let B_j, $0 \le j \le k - 1$, denote a block of size $2N \times 2N$. Note that B_j consists of $PE(u, 2Nj + w)$, $0 \le j$, $w \le 2N - 1$. B_j computes $C_{j+1} = (S_j + C_j)\ div\ 2$, where S_j is given in 2UN representation as $x_{0,j}, x_{1,j}, \ldots, x_{N-1,j}$. C_j is represented in 1UN format. Initially, S_j is stored in $PE(0, u)$, $0 \le u \le N - 1$ in block j. C_j is stored in $PE(u, 0)$, $0 \le u \le N - 1$. The left half (of size $2N \times N$) of the block computes $(S_j + C_j)$ in $O(1)$ time. The technique to convert a 2UN representation into its 1UN representation (Lemma 1) is used for this computation. The output in 1UN representation is available in

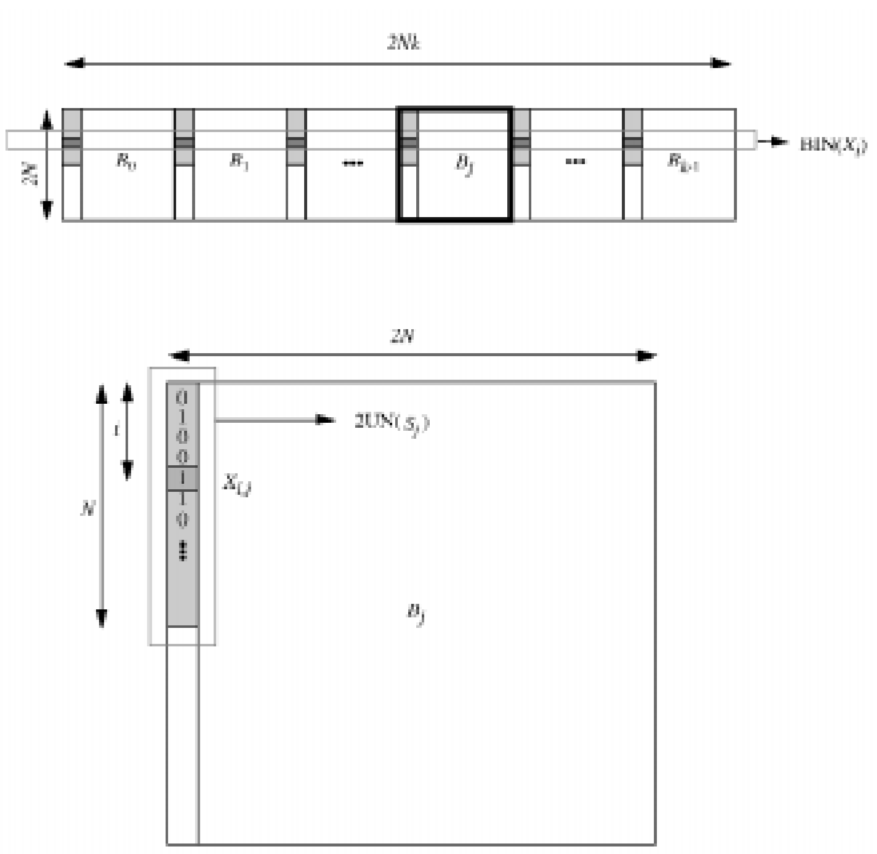

Fig. 5. Initial distribution of the inputs used in Lemma 2.

column $N - 1$. The right half (size $2N \times N$) receives the ($S_j + C_j$) as input and outputs ($S_j + C_j$) *div* 2 in 1UN representation. The right half of a block consisting of PEs in column N through $2N - 1$ is configured such that $PE(u, N)$ is connected to $PE(u/2, 2N - 1)$ only if u is even, $0 \le u \le N - 1$. Note that such a connection realizes the *div* 2 function when the input is provided in 1UN representation. For this, the PEs in the right half of a block are configured as follows:

- {EW, S, N} if $u \le w - N$,
- {NW, SE} if $u > w - N$.

As a result, 1 – *signal* in $PE(u, N)$, u even, is propagated to $PE(u/2, 2N - 1)$. For example, if column N has 1111111100 (7 in 1UN), then the output in column $2N - 1$ would be 1111000000 (3 in 1UN).

Fig. 6 shows an example where $C_j = 3$, $S_j = 2$. Initially, $S_j(101)$ is in $PE(0, 0)$, $PE(1, 0)$, $PE(2, 0)$. It is first routed to $PE(0, 0)$, $PE(0, 1)$, $PE(0, 2)$. $PE(0, 0)$ and $PE(0, 2)$ broadcast *1-signal* while $PE(1, 0)$ broadcasts *0-signal* along its column by configuring $PE(u, w)$, $0 \le u \le 5$, $0 \le w \le 2$ to {NS, W, E}. Thus, $PE(u, 0)$ and $PE(u, 2)$, $0 \le u \le 5$, have *1-signal* and $PE(u, 1)$, $0 \le u \le 5$, have *0-signal*. PEs receiving *1-signal* set their configuration to {SW, NE} while the PEs receiving *0-signal* set their configuration to {EW, N, S} as shown in Fig. 6. $PE(u, w)$, $0 \le u \le 5$, $3 \le w \le 5$ implement the *div* 2 function.

Now, C_j is applied at the leftmost column (i.e., at the W port of the PEs in the leftmost column.) In this example, *1-signal* is input at the W port of $PE(u, 0)$, $0 \le u \le 3$, while *0-signal* is input at the W port of $PE(u, 0)$, $4 \le u \le 5$. S_j in 2UN is input at the N port of $PE(0, u)$, $0 \le u \le 2$. Then, C_{j+1} in 1UN representation is available in $PE(u, 5)$, $0 \le u \le 5$. Thus, $C_{j+1} = (C_j + S_j)$ *div* 2 can be computed in $O(1)$ time using a $2N \times 2N$ reconfigurable mesh.

Let B_j be the $2N \times 2N$ reconfigurable mesh configured to compute $C_{j+1} = (C_j + S_j)$ *div* 2. B_j, $0 \le j \le k - 1$ are cascaded to compute C_{j+1}, $0 \le j \le k - 1$ in parallel. Inputs to the cascade are S_j, $0 \le j \le k - 1$ and $C_0(= 0)$. Then, C_{j+1} in 1UN representation is available at the right most column of B_j, for $0 \le j \le k - 1$.

Now we describe how we can obtain output $z_j = (C_j + S_j)$ *mod* 2. In Fig. 6, ($C_j + S_j$) in 1UN representation is available in $PE(u, 2)$, $0 \le u \le 5$. Assume that $PE(u, N - 1)$ knows if u is even (1 bit is sufficient to represent this information).

If $C_j + S_j = r_0$, then $PE(r_0, 2)$ becomes active while all other PEs are inactive. This can be done in parallel by checking adjacent PEs along column $N - 1$. If r_0 is odd, then $PE(r_0, N - 1)$ broadcasts *1-signal* along its

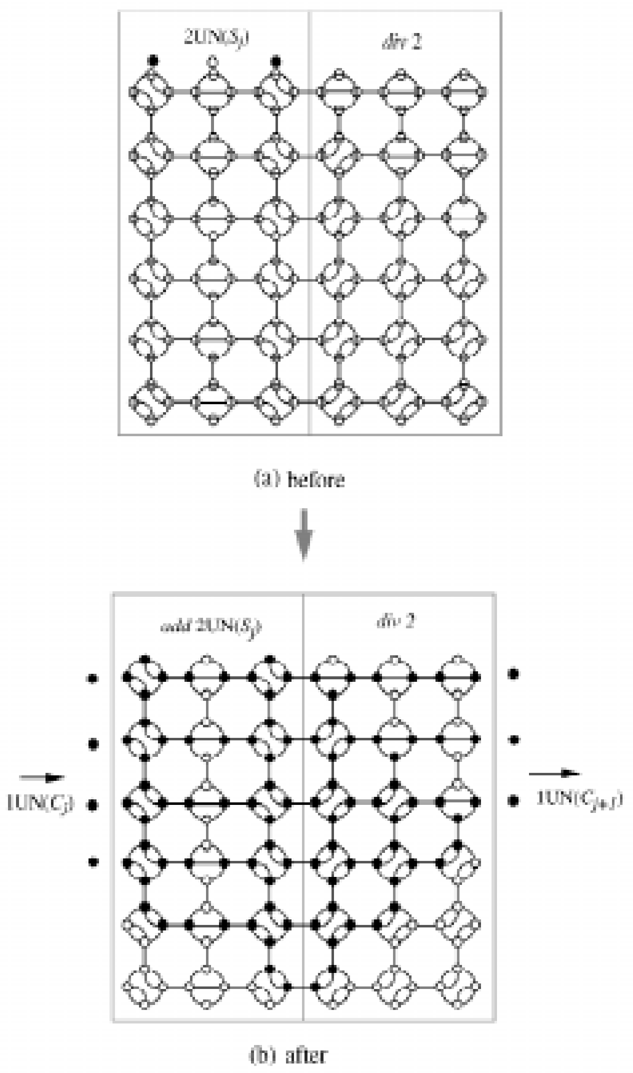

Fig. 6. Computation of $C_{j+1} = (C_j + S_j)$ *div* 2 for $C_j = 3$ and $S_j = 2$.

column, otherwise it broadcasts *0-signal*. $PE(0, N-1)$ will have either *1-signal* or *0-signal* on its N port. This represents $z_j = (C_j + S_j)$ *mod* 2. Now z_j, $0 \le j \le k-1$ can be routed in $O(1)$ time to $PE(0, j)$, $0 \le j \le k-1$.

By setting $S_j = 0$ for $k \le j \le k - 1 + \log N$, z_j for $k \le j \le k - 1 + \log N$ can be easily determined once C_k is known. z_j, $0 \le j \le k - 1 + \log N$ form the BIN representation of the output. □

3 The Multiplication Algorithm

Given two N bit binary numbers, $X = x_0 + x_1 2 + x_2 2^2 + \ldots + x_{N-1} 2^{N-1}$, $Y = y_0 + y_1 2 + y_2 2^2 + \ldots + y_{N-1} 2^{N-1}$, the *multiplication problem* is to compute the $2N$-bit product of X and Y. This can be performed by computing $\sum_{i=0}^{N-1} X \times y_i 2^i$. For ease of explanation, we assume N is a power of 2. Wang, Chen and Li [31] solve this problem in $O(1)$ time on an $N^2 \times N^2$ reconfigurable mesh. Their approach generates the N $2N$-bit product terms and sums them up using parallel computation of carries. They used a large reconfigurable mesh to avoid conflicts of bus signals while computing the carries. Such conflicts occur due to backward propagation of signals over buses when blocks computing carries are cascaded. Note that Lemma 2 leads to:

Corollary 1. *Multiplication of two N-bit numbers can be performed in $O(1)$ time on a $N \times N^2$ reconfigurable mesh.*

First, we briefly review Rader Transform (RT) which is used in our algorithm to perform cyclic convolution.

3.1 Rader Transform

It is known that RT, which is a transform in a finite field (and more generally in a ring), has *cyclic convolution property* and is without roundoff error [23]. The ring is closed under addition and multiplication modulo some integer F_t of the form 2^{2^t+1}. Given input $x(n)$, $0 \le n \le N-1$, where $x(n)$ is an element in the ring, the RT given by:

$$X(k) = \sum_{n=0}^{N-1} x(n)\alpha^{nk} \bmod(F_t), \quad k = 0, 1, \ldots, N-1. \quad (1)$$

The inverse RT is defined as:

$$x(n) = 2^{-m} \sum_{k=0}^{N-1} X(k)\alpha^{-nk} \bmod(F_t), n = 0, 1, \ldots, N-1. \quad (2)$$

Note that

$$F_t = 2^{2^t} + 1, \quad \text{for some integer } t, \text{ and } N \le 2^{t+1}. \quad (3)$$

N is the least positive integer such that $\alpha^N = 1 \bmod(F_t)$ and α is a power of 2 and is an Nth root of 1 in the ring. Note that $\alpha^{-1} = \alpha^{N-1}$ *mod* F_t and $2^{-m} = -2^{2^t-m}$ *mod* F_t.

3.1.1 Diminished-1 Notation

In computing RT in the ring of integers modulo $2^B + 1$, arithmetic modulo $2^B + 1$ is performed. $B + 1$ bits are needed to represent a number in the ring. If the result r from an arithmetic operation is greater than 2^B, r *mod* $(2^B + 1)$ should be computed to generate a valid output. To avoid this extra modular computation, Agarwal and Burrus [2] limit their realization to B-bit arithmetic in which only those operands in the range of $[0, 2^B - 1]$ are correctly represented. This involves some quantization error when 2^B occurs as an operand. In order to overcome this problem, we employ the *diminished-1 notation* [12]. By using this notation, we can perform modular arithmetic as simple as in [2] without any quantization error. In the *diminished-1 notation*, number i, $1 \le i \le 2^B$, is represented by the binary representation of number $i - 1$ and 0 is represented by 2^B. Thus, 0 is represented by a 1 in the $(B+1)$th bit position. The result of an addition and multiply operation, where one of the operands is zero, can be obtained without actual computation. First, check if one of the operands is zero by inspecting the $(B + 1)$th bit. Note that the bits are numbered 1 to $B + 1$, starting from the right-most bit. If so, output the result without any further computation. If not, since both operands can be represented using only least significant B bits, perform the desired computation using B bits. Using this approach, we can show:

LEMMA 3. *Given numbers in the ring of integers modulo* $2^B + 1$ *represented using the diminished-1 notation, addition of two numbers or multiplication by a constant* 2^k *in the ring of integers modulo* $2^B + 1$, $0 \le k \le B - 1$, *can be performed in* $O(1)$ *time on a* $B \times B$ *reconfigurable mesh.*

PROOF. For ease of explanation we use a $(B + 1) \times (B + 1)$ mesh. Let X, Y, Z be integers represented in the *diminished-1 notation* using $B + 1$ bits. Let k be a constant such that $0 \le k \le B - 1$.

- $X + Y$ modulo $(2^B + 1)$:
  If the $(B + 1)$th bit of either X or Y is 1, inhibit the addition and the other addend is the desired sum. If the $(B + 1)$th bit of both X and Y are 0, ignore the $(B + 1)$th bit and add the remaining B bits of each operand. Complement the carry and add it to the least significant B bits of the sum. For example, suppose $B = 4$, $X = 4$, and $Y = 6$. In the *diminished-1 notation*, $X = 00011$ and $Y = 00101$. Addition of 0011 and 0101 gives carry 0 and sum 1000. Complementing the carry and adding it to the sum gives 10(01001). The above processing can be performed in $O(1)$ time on $1 \times (B + 1)$ reconfigurable mesh by using the technique shown in Fig. 4.
- $Z \times 2^k$ modulo $(2^B + 1)$:
  Initially, the $(B + 1)$ bits of Z are assumed to be in $PE(0, j)$, $0 \le j \le B$. If the $(B + 1)$th bit of Z is 1, the multiplication is inhibited and the result is zero. Otherwise, left shift Z by k bits ignoring the $(B + 1)$th bit and add the complement of the k shifted-out bits. For example, suppose $B = 4$, $Z = 11$ and $k = 3$. In the *diminished-1 notation*, Z is represented as 01010. Left shift of 1010 by 3 bits gives 00000. Complementing the three shifted-out bits (101) and adding it to 00000 results in 00010 which represents three in the *diminished-1 notation*. Note that 11×2^3 modulo $2^4 + 1 = 3$. Since any permutation of $(B + 1)$ bits stored in $PE(0, j)$, $0 \le j \le B$ can be performed in $O(1)$ time on a $(B + 1) \times (B + 1)$ reconfigurable mesh, the above processing can be performed in $O(1)$ time. □

Combining Lemma 2 and Lemma 3, we have:

LEMMA 4. *Given* $2P$ *points in the BIN representation in a row, the* $2P$*-point RT (and its inverse) in the ring of integers modulo* $2^B + 1$ *can be performed in* $O(1)$ *time on a* $P^2B \times PB$ *reconfigurable mesh, where* $P \le B$ *and* B *is a multiple of* P *and a power of* 2.

PROOF. Let α be the $2P$th primitive root of unity, i.e., $2P$ is the smallest integer such that $\alpha^{2P} = 1 \ mod \ (2^B + 1)$. Note that $2^{2B} \ mod \ (2^B + 1) = 1$. Thus, $\alpha = 2^{2B/2P}$. The corresponding transform matrix is:

$$\mathbf{T} = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \alpha & \alpha^2 & \dots & \alpha^{2P-1} \\ 1 & \alpha^2 & \alpha^4 & \dots & \alpha^{4P-2} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \alpha^{2P-1} & \alpha^{4P-2} & \dots & \alpha^{(2P-1)(2P-1)} \end{bmatrix}$$

Let $\mathbf{x}$ denote an input vector of $2P$ integers:

$$\mathbf{x} = \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ \vdots \\ x(2P-1) \end{bmatrix}$$

Similarly, let $\mathbf{X}$ denote the RT of $\mathbf{x}$. Note that $\mathbf{X}$ is a vector of $2P$ integers. Then, $\mathbf{X} = \mathbf{Tx}$ computed using arithmetic in the ring of integers modulo $2^B + 1$. Direct computation of $\mathbf{Tx}$ would require $4P^2$ multiplications of $(B + 1)$-bit operands and $2P(2P - 1)$ additions. For ease of explanation, we will use a $4P^2B \times 2PB$ reconfigurable mesh. Initially, the BIN representation of $x(j)$ is stored in $PE(0, jB + w)$, $0 \le w \le B-1$ as shown in Fig. 7a. Partition the mesh into $RM(i)$, $0 \le i \le 2P - 1$. $RM(i)$ is a submesh of size $2PB \times 2PB$ which consists of $PE(i2PB + u, w)$, $0 \le u, w \le 2PB - 1$. We will show that $X(i) = \sum_{j=0}^{2P-1} \alpha^{ij} x(j) \ mod \left(2^B + 1\right)$ can be computed in $RM(i)$ in $O(1)$ time. $x(j)$, $0 \le j \le 2P - 1$ (a total of $2PB$ bits) is broadcast to the $2PB$ PEs in the top row of each $RM(i)$, $0 \le i \le 2P - 1$. $RM(i)$ is subdivided into $BM(i,j)$, $0 \le j \le 2P - 1$. $BM(i,j)$ is of size $2PB \times B$ and consists of $PE(i2PB + u, jB + w)$, $0 \le u \le 2PB - 1$, $0 \le w \le B - 1$ (see Fig. 7b). At this time, $x(j)$ is in the top row of $B(i, j)$.

Observe that $\alpha^{ij}x(j) \ mod \ (2^B + 1)$ can be computed in $BM(i, j)$ in $O(1)$ time. Since $\alpha^{2P} = 1$, α^{ij} can be replaced by α^k, where, $0 \le k \le 2P - 1$. Since $\alpha = 2^{2B/2P}$, for any k, $0 \le k \le 2P - 1$, $\alpha^k = 2^{k'}$ where $0 \le k' \le 2B - 1$. k' can be determined a priori (independent of input $x(j)$).

Now, $x(j)$ in BIN representation is converted into *diminished-1 notation* and multiplied by $2^{k'}$ in modulo $(2^B + 1)$ arithmetic using Lemma 3. The resulting $\alpha^{ij}x(j) \ mod \ (2^B + 1)$ in *diminished-1 notation* is converted into BIN representation. Recall that $0 \le \alpha^{ij}x(j) \ mod \ (2^B + 1) \le 2^B$. Actually, we need $B + 1$ PEs to represent $\alpha^{ij}x(j) \ mod \ (2^B + 1)$ in BIN representation. But this can be handled without increasing the asymptotic time complexity. For ease of explanation, assume that it is represented by B bits. Let $x'(j)$ denote $\alpha^{ij}x(j) \ mod \ (2^B + 1)$ in BIN representation. This is computed in $BM(i, j)$. Then, the B bits of $x'(j)$ are routed to $PE(i2PB$

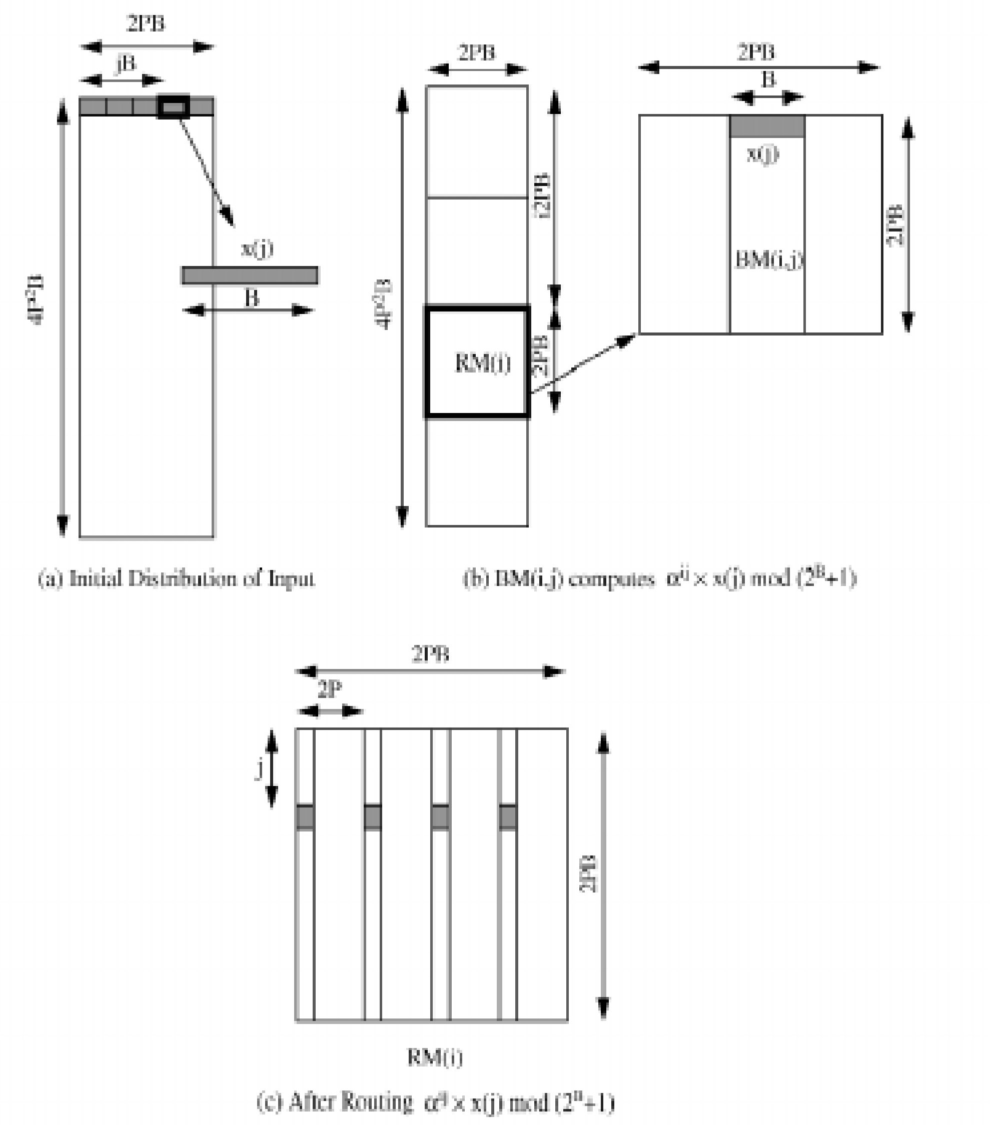
2PB
jB
x(j)
B
4P²B
(a) Initial Distribution of Input
2PB
i2PB
2PB
RM(i)
4P²B
2PB
B
x(j)
BM(i,j)
2PB
(b) BM(i,j) computes α^{ij} × x(j) mod (2^B+1)
2PB
2P
j
2PB
RM(i)
(c) After Routing α^{ij} × x(j) mod (2^B+1)

Fig. 7. Data movement in the proof of Lemma 4.

$+ j, w2P)$, $0 \le w \le B - 1$ (see Fig. 7c). This is possible since $RM(i)$ is of size $2PB \times 2PB$ and $x'(j)$, $0 \le j \le 2P - 1$ are stored in the top row of $RM(i)$. Note that the total number of bits to be routed is 2PB. Now the summation of $x'(j)$, $0 \le j \le 2P - 1$ can be performed in $O(1)$ time using Lemma 2. The result is in BIN representation. Computing $X(i)$ modulo $(2^B + 1)$ can be performed in $O(1)$ time using $RM(i)$, $0 \le i \le 2P - 1$. □

3.2 Outline of the Multiplication Algorithm

Let C_i, $0 \le i \le 2P - 1$ be the cyclic convolution of two sequences A_i, B_i, $0 \le i \le 2P - 1$. Then C_i is defined by:

$$C_i = \sum_{k=0}^{2P-1} A_{i-k} B_k, \quad 0 \le i \le 2P - 1$$

where A_i, B_i are periodically extended outside their original domain.

Multiplication of two N-bit integers is reducible to the cyclic convolution of two integer sequences of length $2P$ with each integer in $[0, 2^{N/P} - 1]$. We choose P to be $N^{3/4}$, the reason for which will become clear during the course of the discussion. The cyclic convolution of two integer sequences can be performed by the following three steps [2].

1) Compute the *transform* of each sequence.
2) Compute the product of the two *transforms*. (i.e., multiply the corresponding elements.)
3) Obtain the inverse *transform* of the product.

where *transform* is any transform satisfying *cyclic convolution property*. A transform is said to have the *cyclic convolution property* iff the transform of the cyclic convolution of the two sequences is equal to the product of the transforms of the two individual sequences. To avoid round-off errors, the *transform* is computed in a finite field of integers. The difficulty in using a finite field is in finding a suitable field with an Nth root of unity such that the size of the transform is proportional to the size of the finite field. If we can employ such a field, the transform can be computed using words of length proportional to $\log N$.

We choose $2N^{3/4}$-point Rader Transform(RT) for the *transform* in Step 1 above. As discussed earlier, N-point RT is a transform in a finite field of integers modulo an integer of the form $2^B + 1$ and has a power of 2 as the kernel (or Nth root of unity), where $N/2 \le B$ [23]. RT is chosen in spite of its demand for larger word length requirement ($O(N)$) because of the following two merits. First, there is a systematic and simple way to find a finite field which has an Nth root of unity. Second, since every element of the transform matrix is a power of 2, multiplication of an input integer by an element of the matrix can be reduced to shift operations. To perform arithmetic modulo $2^B + 1$ efficiently, *diminished-1 notation* [12] is used. Lemma 3 (in Section 3.2) shows how addition of two numbers and multiplication by a power of 2 in the ring of integers modulo $2^B + 1$ can be performed on the reconfigurable mesh. In the proof of Lemma 4, detailed data movement is shown for the computation of $2P$-point RT in the ring of integers modulo $2^B + 1$ in $O(1)$ time on $P^2B \times PB$ reconfigurable mesh. Using Lemma 4, computation of $2N^{3/4}$-point RT in $O(1)$ time would require a mesh of size $N^{9/4} \times N^{6/4}$. Reducing the *multiplication problem* to convolution of $2\sqrt{N}$ integers in $\left[0, 2^{N^{1/2}} - 1\right]$ (i.e., choosing $P = \sqrt{N}$ instead of $N^{3/4}$), does not work either. The problem is with the summation of resulting $\sqrt{N}$ numbers($O(N)$ bits in total) to obtain an element of the output vector. In order to use Lemma 2, these $O(N)$ bits should be permuted in $O(1)$ time on an $\sqrt{N} \times N$ mesh to obtain the initial input distribution shown in Fig. 5. This is impossible due to the narrow bandwidth of the mesh.

To avoid the above problems, we decompose the $2N^{3/4}$-point (one-dimensional) cyclic convolution into four-dimensional cyclic convolution, where the convolution in each dimension is performed on $8N^{9/16}$ groups of $2N^{3/16}$ points($8N^{9/16}$ convolutions with each convolution on $2N^{3/16}$ points). The reason we have chosen $P = N^{3/4}$ is that word length of $N/P(= N^{1/4})$ is large enough to compute each $2N^{3/16}$-point convolution. The details are shown in Lemma 6. First, we begin with computing the cyclic convolution.

LEMMA 5. *The cyclic convolution of two integer sequences of length* $2P$ *with each integer in* $[0, 2^B - 1]$ *represented using the BIN notation can be performed in* $O(1)$ *time on a* $P^2B \times PB$ *reconfigurable mesh, where* $P \le B \le P^2$ *and* B *is a* mul*tiple of* P *and is a power of* 2.

PROOF. Let C_i, $0 \le i \le 2P - 1$ be the cyclic convolution of two sequences A_i, B_i, $0 \le i \le 2P - 1$. Then C_i is defined by:

$$C_i = \sum_{k=0}^{2P-1} A_{i-k} B_k, \quad 0 \le i \le 2P - 1$$

where A_i, B_i are periodically extended outside their original domain. Since RT has the *cyclic convolution property*, the cyclic convolution can be computed using RT. Since $A_i, B_i \in [0, 2^B - 1]$, it is clear that $C_i \in [0, 2^{3B-1} - 1]$. In the ring of integers modulo $2^{3B} + 1$, when two integer sequences A_i and B_i are convolved, the output integer sequence C_i is congruent to the convolution of A_i and B_i modulo $2^{3B} + 1$. In the ring of integers modulo $2^{3B} + 1$, conventional integers can be represented unambiguously if their absolute value is less than $(2^{3B} + 1)/2$. Since the input sequences A_i, B_i are scaled such that C_i never exceeds $(2^{3B} + 1)/2$, we would get the same result by implementing the convolution in the ring of integers modulo $2^{3B} + 1$ as that obtained with the usual arithmetic. Based on this and the cyclic convolution property of RT, C_i can be obtained by the following three steps:

Step 1. Compute RT of A_i, B_i in the ring of integers modulo 2^{3B}+1.
Step 2. Compute the product of the two RTs.
Step 3. Compute the inverse RT of the product.

From Lemma 4, Step 1 can be performed in $O(1)$ time on a $3P^2B \times 3PB$ mesh. $2P$ multiplications of $3B$-bit numbers are needed in Step 2. From Corollary 1, multiplication of two $3B$-bit numbers can be performed in $O(1)$ time on a $9B^2 \times 3B$ mesh. Since $P^2 \ge B$, Step 2 can be performed in $O(1)$ time on a $9P^2B \times 6PB$ mesh. Step 3 can be performed similar to Step 1. □

It is known that one-dimensional cyclic convolution of two sequences of P numbers can be replaced by d dimensional cyclic convolution of two $\overbrace{D \times D \times \ldots \times D}^{d \text{ times}}$ arrays, where $D = (2^{d-1}P)^{1/d}$. The $2^{d-1}P$ numbers are the original P numbers padded with 0s to allow cyclic convolution along each of the d dimensions. For details, refer to [1]. For ease of explanation, we assume $(2^{d-1}P)^{1/d}$ to be an integer. Cyclic convolution along each dimension can be performed by $(2^{d-1}P)^{1-1/d}$ cyclic convolutions with each convolution performed on two sequences of $(2^{d-1}P)^{1/d}$ numbers. Based on this and Lemma 5, we have:

LEMMA 6. *Cyclic convolution of two integer sequences of length* $2N^{3/4}$ *where each integer is in* $\left[0, 2^{N^{1/4}} - 1\right]$ *can be performed in* $O(1)$ *time on an* $N \times N$ *reconfigurable mesh.*

PROOF. Choosing $d = 4$ in the above discussion, the $2N^{3/4}$-point cyclic convolution can be replaced by $(16N^{3/4})^{3/4}$ cyclic convolutions along each dimension with each convolution performed on two sequences of $(16N^{3/4})^{1/4}$ integers. We will use a $16N \times 16N$ mesh. Let A_j, B_j, $0 \le j \le 2N^{3/4} - 1$ be the two sequences to be convolved. Initially, A_j, B_j are in $PE(0, jN^{1/4} + w)$, $0 \le w$

$\le N^{1/4} - 1$, $0 \le j \le N^{3/4} - 1$. By appending the $2N^{3/4}$ input points with 0s for performing four-dimensional convolution, we have a (conceptual) four-dimensional data array of size $2N^{3/16} \times 2N^{3/16} \times 2N^{3/16} \times 2N^{3/16}$. These 4D data arrays generated from A_j and B_j are stored in lexicographic order in the top row of the $16N \times 16N$ mesh. For example, data in PEs at a distance of $2N^{3/16}$ in the top row are adjacent along the second dimension of the 4D data array. Divide the mesh into vertical submeshes of size $16N \times 2N^{3/16}N^{1/4}$. Then, each submesh has $2N^{3/16}$ numbers (from A_j and B_j) which are adjacent along the first dimension of the 4D data array. From Lemma 5, each convolution on two sequences of length $2N^{3/16}$ can be performed in $O(1)$ time in a submesh. Thus, $(16N^{3/4})^{3/4}$ cyclic convolutions along the first dimension can be performed in parallel in $O(1)$ time on a $16N \times 16N$ mesh. Now the resulting numbers in the top row are permuted in such a way that the numbers adjacent to each other along the second dimension of the 4D data array are adjacent in the top row of the $16N \times 16N$ mesh. $16N$ bits are permuted and this can be performed in $O(1)$ time on a $16N \times 16N$ mesh. Using the above idea for each of the four dimensions, the convolution can be completed in $O(1)$ time. □

Now we are ready for the proof of the main theorem:

THEOREM 1. *Multiplication of two N bit numbers given in a row can be performed in* $O(1)$ *time on an* $N \times N$ *reconfigurable mesh.*

PROOF. Multiplication of two numbers, $a_{N-1}a_{N-2} \dots a_0$, $b_{N-1}b_{N-2} \dots b_0$ can be represented by $C\left(2^{N^{1/4}}\right)$ where

$$A(x) = \sum_{i=0}^{2N^{3/4}-1} A_i x^i \tag{4}$$

$$A_i = a_{(i+1)N^{1/4}-1}\, a_{(i+1)N^{1/4}-2} \dots a_{iN^{1/4}},\ 0 \le i \le N^{3/4} - 1 \tag{5}$$

$$A_i = 0,\ N^{3/4} \le i \le 2N^{3/4} - 1 \tag{6}$$

$$B(x) = \sum_{i=0}^{2N^{3/4}-1} B_i x^i \tag{7}$$

$$B_i = b_{(i+1)N^{1/4}-1}\, b_{(i+1)N^{1/4}-2} \dots b_{iN^{1/4}},\ 0 \le i \le N^{3/4} - 1 \tag{8}$$

$$B_i = 0,\ N^{3/4} \le i \le 2N^{3/4} - 1 \tag{9}$$

$$C(x) = \sum_{i=0}^{2N^{3/4}-1} C_i x^i \tag{10}$$

$$C_i = \sum_{k=0}^{i} A_k B_{i-k},\ 0 \le i \le 2N^{3/4} - 1 \tag{11}$$

Clearly, $A_i, B_i \in \left[0, 2^{N^{1/4}} - 1\right]$ and $C_i \in \left[0, 2^{3N^{1/4}-1} - 1\right]$ for all N s. t. $N^{1/4} - 1 \ge \log N^{3/4}$. If we can compute C_i, $0 \le i \le 2N^{3/4} - 1$ in $O(1)$ time, then we can obtain the result by computing $C\left(2^{N^{1/4}}\right)$. C_i, $0 \le i \le 2N^{3/4} - 1$ can be obtained by cyclic convolution of A_i and B_i. From Lemma 6, this can be performed in $O(1)$ time on an $N \times N$ mesh. Now, we have C_i, $0 \le i \le 2N^{3/4}–1$. The desired output is $\sum_{i=0}^{2N^{3/4}-1} C_i 2^{iN^{1/4}}$. Let $N' = \lfloor N^{3/4}/3 \rfloor$, then, we have:

$$\sum_{i=0}^{2N^{3/4}-1} C_i 2^{iN^{1/4}} = \sum_{l=0}^{N'-1} C_{3l} 2^{3lN^{1/4}} + \sum_{l=0}^{N'-1} C_{3l+1} 2^{3lN^{1/4}+1} + \sum_{l=0}^{N'-1} C_{3l+2} 2^{3lN^{1/4}+2}$$

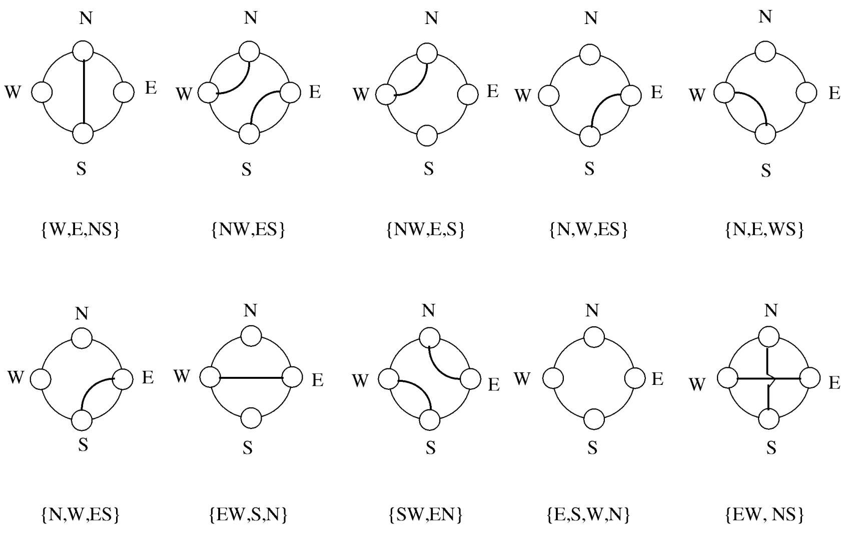

Fig. 8. Connection patterns allowed in MRN/LRN.

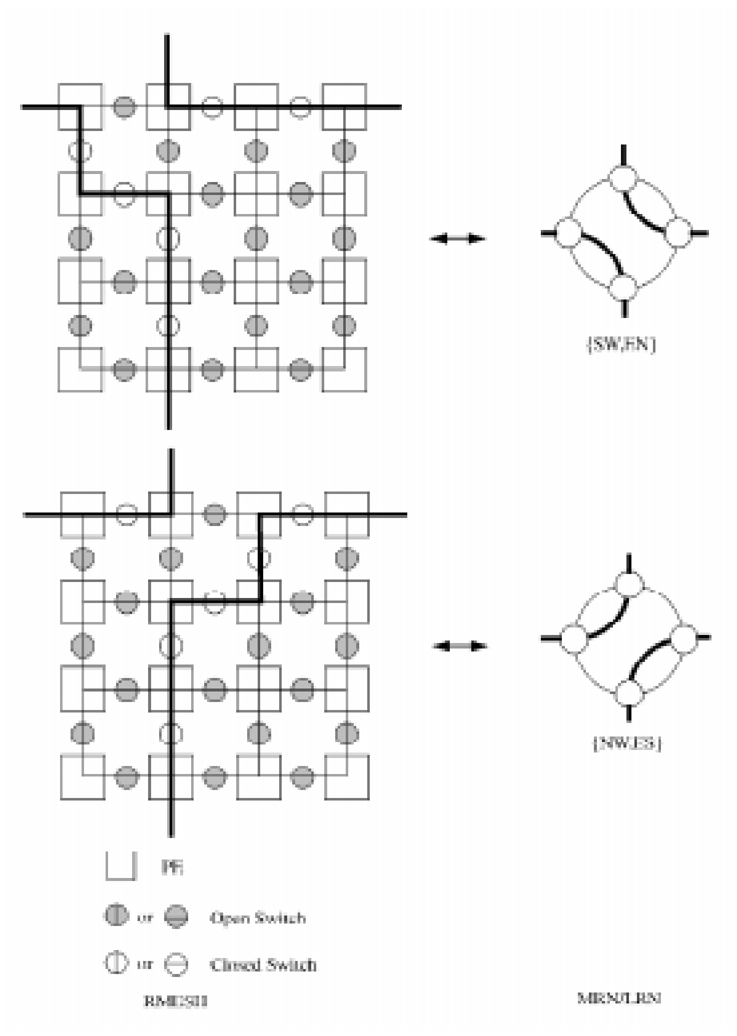

Fig. 9. Simulation of some patterns of MRN/LRN on RMESH.

Note that $\sum_{l=0}^{N'-1} C_{3l} 2^{3lN^{1/4}}$ can be obtained by concatenation of C_{3l}, $0 \le l \le N' - 1$. The desired output can be obtained by adding the three $2N$-bit numbers obtained by concatenation of C_{3l}, C_{3l+1}, C_{3l+2}. This addition can be performed in $O(1)$ time on an $N \times N$ reconfigurable mesh. □

3.3 Area-Time Trade-off

It is known [5] that for the multiplication of two N-bit numbers, $AT^{1+\alpha} = \Omega(N^{1+\alpha})$, for $0 \le \alpha \le 1$, in the two-dimensional bit-model of VLSI (details of the bit-model of VLSI can be found in [28]). In the bit-model of reconfigurable mesh, each PE has $O(1)$-bit memory and $O(1)$ area control unit which can store $O(1)$ patterns of switch settings. In addition, the ALU in each PE occupies $O(1)$ area. Thus, each PE has $O(1)$ area. In each PE, an arithmetic/logic operation is performed in unit time.

Note that our model allows the use of the reconfigurable connections. The standard bit-model of VLSI (which was defined before the reconfigurable mesh was introduced), did not contemplate the use of reconfigurable connections. However, the above lower bound for multiplication is based on information transfer arguments and hence it is true in the model considered here.

Known VLSI designs satisfying the AT^2 lower bound fall into the range $\log N \le T \le \sqrt{N}$ [16]. In this section, Theorem 1 is extended to show a VLSI design which satisfies AT^2 optimality over $1 \le T \le \sqrt{N}$.

THEOREM 2. *There is a VLSI design to multiply two N-bit numbers satisfying $AT^2 = O(N^2)$, over $1 \le T \le \sqrt{N}$.*

PROOF. Multiplication of two N-bit numbers can be reduced to cyclic convolution of two integer sequences of length $2N^{5/6}$ with each integer in $\left[0, 2^{N^{1/6}} - 1\right]$. The cyclic convolution can be performed by $(64N^{5/6})^{5/6}$ cyclic convolutions along each dimension. Note that each convolution is performed on two sequences of $(64N^{5/6})^{1/6}$ integers. We will show that $(64N^{5/6})^{5/6}$ cyclic convolutions along six dimensions can be performed in $O(T)$ time on a $N/T \times N/T$ reconfigurable mesh. Without loss of generality, assume $kT = N^{1/2}$ for some integer k. Then, $N/T = kN^{1/2}$. We will use a $64N/T \times 64N/T$ mesh. The 6D data array can be unfolded as in Lemma 6 and stored in T consecutive rows. Processing is performed row by row. Divide the $64N/T \times 64N/T$ mesh into blocks of size $64N/T \times 2N^{5/36}N^{1/6}$. From Lemma 5, each convolution on two sequences of $(64N^{5/6})^{1/6}$ integers can be performed in $O(1)$ time in each block. Since we have $32kN^{7/36}$ such blocks in the $64N/T \times 64N/T$ mesh, $32kN^{7/36}$ convolutions can be performed in $O(1)$ time. Since $32kN^{7/36} \times T = 32kTN^{7/36} = 32N^{1/2}N^{7/36} = (64N^{5/6})^{5/6}$, the $(64N^{5/6})^{5/6}$ cyclic convolutions along each dimension can be performed in $O(T)$ time on a $64N/T \times 64N/T$ mesh. Permutation of $64N$ bits in preparation for cyclic convolution along the next dimension can be performed in T steps, where in each step, $64N/T$ bits are permuted in $O(1)$ time using a $64N/T \times 64N/T$ mesh. □

3.4 Multiplication on Related Models

After the definition of the reconfigurable mesh model in [17], other models have been defined [3], [20], [30]. These models restrict the allowed connection patterns. The most general (and the most powerful) one is the PARBUS model [30] which allows any combination of connections among the four ports in each PE. Notice that the PARBUS bit-model is same as our model in Section 2.1. In [3], the Reconfigurable Network (RN) model has been introduced. Several algorithms on this model have been shown under the mesh restriction. This model has been denoted as MRN in [20] and as LRN in [4]. The connection patterns allowed by MRN/LRN are shown in Fig. 8. In MRN, the number of possible connection patterns within each PE is 10. In [10], the RMESH is introduced, which doesn't allow the {EW, NS}, {NE, SW}, and the {NW, SE} connections that are allowed in MRN. However, the RMESH allows {NEWS}, {NEW, S}, {NES, W}, {NWS, E}, and {N, EWS} connections. Thus, the total number of possible connection patterns in each PE of the RMESH is 12. This corresponds to having switches on mesh links only [17].

Our multiplication algorithm can be simulated on the MRN without slowdown, since we only employ a subset of the connection patterns shown in Fig. 8. Our multiplication algorithm can also be simulated on the RMESH. All the connections patterns shown in Fig. 8 except {EW, NS} can be simulated by a constant number of PEs in the RMESH. Fig. 9 shows an example illustration of such a simulation. Other patterns can be simulated in a similar way. Since our multiplication algorithm does not employ the {EW, NS} pattern, it can also be simulated on the RMESH model without *asymptotic* increase in the time complexity or in the number of PEs employed.

4 Conclusion

It is shown that multiplication of two N-bit integers can be performed in $O(1)$ time on an $N \times N$ bit-model of reconfigurable mesh. Previously known $O(1)$ time result [31] uses $N^2 \times N^2$ mesh. Our result is obtained by combining our $O(1)$ time multiplication algorithm on $N \times N^2$ reconfigurable mesh, Rader Transform, and decomposition of one-dimensional convolution into a multidimensional convolution. Choosing Rader transform at the expense of long word length frees us from storing twiddle factors in advance, which is needed in other designs for multiplication [5], [16], [22]. It is also shown that our algorithm can be simulated on other (restricted) reconfigurable mesh models without *asymptotic* increase in time or the number of PEs used.

Acknowledgements

We would like to thank Neungsoo Park for his assistance in preparing the final version of the paper.

This research was supported in part by the U.S. National Science Foundation under grant IRI-9145810 and in part by the U.S. Defense Advanced Research Projects Agency and the U.S. Air Force Office of Scientific Research under contracts F-49620-89-C-0126 and F49620-90-C-0078 and in part by the Korea Science and Engineering Foundation under contract 961-0909-052-1.

References

[1] R.C. Agarwal and C.S. Burrus, "Fast One-Dimensional Digital Convolution by Multidimensional Techniques," *IEEE Trans. Acoustics, Speech, and Signal Processing,* vol. 22, no. 1, pp. 1–10, Feb. 1974.

[2] R.C. Agarwal and C.S. Burrus, "Fast Convolution Using Fermat Number Transforms with Applications to Digital Filtering," *IEEE Trans. Acoustics, Speech, and Signal Processing,* vol. 22, no. 2, pp. 87–92, Apr. 1974.

[3] Y. Ben-Asher, D. Peleg, R. Ramaswami, and A. Schuster, "The Power of Reconfiguration," *J. Parallel and Distributed Computing,* vol. 13, no. 2, pp. 139–153, 1991.

[4] Y. Ben-Asher, D. Gordon, and A. Schuster, "Optimal Simulations in Reconfigurable Arrays," Technical Report #716, Computer Science Dept., Israel Inst. of Technology, Feb. 1992.

[5] R.P. Brent and H.T. Kung, "The Area-Time Complexity of Binary Multiplication," *J. ACM,* vol. 28, no. 3, July 1981.

[6] B. Chazelle and L. Monier, "A Model of Computations for VLSI with Related Complexity Results," technical report, Dept. of Computer Science, Carnegie-Mellon Univ., Feb. 1981.

[7] D.M. Champion and J. Rothstein, "Immediate Parallel Solution of the Longest Common Subsequence Problem," *Proc. Int'l Conf. Parallel Processing,* pp. 70–77, Aug. 1987.

[8] J. Jang and V.K. Prasanna, "An Optimal Sorting Algorithm on Reconfigurable Mesh," *Proc. Int'l Parallel Processing Symp.,* pp. 130–137, Mar. 1992.

[9] J. Jang, H. Park, and V.K. Prasanna, "A Fast Algorithm for Computing Histogram on Reconfigurable Mesh," *Proc. Frontiers of Massively Parallel Computation '92,* pp. 244–251, 1992

[10] J. Jenq and S. Sahni, "Reconfigurable Mesh Algorithms for Image Shrinking, Expanding, Clustering, and Template Matching," *Proc. Int'l Parallel Processing Symp.,* pp. 208–215, 1991.

[11] J. Jenq and S. Sahni, "Reconfigurable Mesh Algorithms for the Area and Perimeter of Image Components," *Proc. Int'l Conf. Parallel Processing,* pp. 280–281, Pennsylvania State Univ. Press, 1991.

[12] L.M. Leibowitz, "A Simplified Binary Arithmetic for the Fermat Number Transform," *IEEE Trans. Acoustics, Speech, and Signal Processing,* vol. 24, pp. 356–359, Oct. 1976.

[13] J. Levison, I. Kuroda, and T. Nishitani, "A Reconfigurable Processor Array with Routing LSIs General Purpose DSPs," *Proc. Int'l Conf. Application Specific Array Processors,* Oct. 1992.

[14] H. Li and M. Maresca, "Polymorphic-Torus Network," *IEEE Trans. Computers,* vol. 38, no. 9, pp. 1,345–1,351, Sept. 1989.

[15] R. Lin, S. Olariu, J. Schwing, and J. Zhang, "Sorting in $O(1)$ Time on an $n \times n$ Reconfigurable Mesh," *Parallel Computing: From Theory to Sound Practice, Proc. of EWPC '92,* pp. 16–27. Amsterdam: IOS Press, 1992.

[16] K. Mehlhorn and F.P. Preparata, “Area-Time Optimal VLSI Integer Multiplier with Minimum Computation Time,” *Information and Control*, vol. 58, pp. 137–156, 1983.

[17] R. Miller, V.K. Prasanna Kumar, D.I. Reisis, and Q.F. Stout, “Meshes with Reconfigurable Buses,” *Proc. Fifth MIT Conf. Advanced Research in VLSI*, pp. 163–178, 1988.

[18] R. Miller, V.K. Prasanna Kumar, D.I. Reisis, and Q.F. Stout, “Parallel Computations on Reconfigurable Mesh,” *IEEE Trans. Computers*, vol. 42, no. 6, pp.678–692, June, 1993.

[19] R. Miller, V.K. Prasanna Kumar, D.I. Reisis, and Q.F. Stout, “Image Computations on Reconfigurable Mesh,” *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pp. 925–930, 1988.

[20] M. Nigam and S. Sahni, “Sorting n Numbers on $n \times n$ Reconfigurable Meshes with Buses,” Technical Report TR-92-04, Univ. of Florida, 1992.

[21] F.P. Preparata, “A Mesh-Connected Area-Time Optimal VLSI Multiplier of Large Integers,” *IEEE Trans. Computers*, vol. 32, no. 2, pp.194–198, Feb. 1983.

[22] F.P. Preparata and J. Vuillemin, “Area-Time Optimal VLSI Networks for Computing Integer Multiplication and Discrete Fourier Transform,” *Proc. ICALP*, pp. 29–40, Haifa, 1981.

[23] C.M. Rader, “Discrete Convolution via Mersenne Transforms,” *IEEE Trans. Computers*, vol. 21, no. 12, pp. 1,269–1,273, Dec. 1972.

[24] D.I. Reisis, “Parallel Computations on Meshes with Static and Reconfigurable Buses,” PhD thesis, University of Southern California, Dept. of EE-Systems, July 1989.

[25] D.I. Reisis, “An Efficient Convex Hull Computation on the Reconfigurable Mesh,” *Proc. Int’l Parallel Processing Symp. ‘92*, pp. 142–145, Mar. 1992.

[26] L. Snyder, “Introduction to the Configurable Highly Parallel Computer,” *Computer*, vol. 15, no. 1, pp. 47–56, Jan. 1982.

[27] Q.F. Stout, “Meshes with Multiple Buses,” *Proc. 27th Foundation of Computer Science*, pp. 264–272, 1986.

[28] C.D. Thompson, “Fourier Transforms in VLSI,” *IEEE Trans. Computers*, vol. 32, no. 11, pp. 1,047–1,057, Nov. 1983.

[29] B.F. Wang and G.H. Chen, “Constant Time Algorithms for the Transitive Closure Problem and Some Related Graph Problems with Reconfigurable Bus Systems,” *IEEE Trans. Parallel and Distributed Systems*, vol. 1, no. 4, pp. 500–507, Apr. 1991.

[30] B.F. Wang, G.H. Chen, and F.C. Lin, “Constant Time Sorting on a Processor Array with a Reconfigurable Bus System,” *Information Processing Letters*, vol. 34, pp. 187–192, 1990.

[31] B.F. Wang, G.H. Chen, and H. Li, “Configurational Computation: A New Computation Method on Processor Arrays with Reconfigurable Bus Systems,” *Proc. Int’l Conf. Parallel Processing*, 1991.

[32] C.C. Weems, S.P. Levitan, A.R. Hanson, E.M. Riseman, J.G. Nash, and D.B. Shu, “The Image Understanding Architecture,” *COINS* Technical Report 87-76, Univ. of Massachusetts at Amherst, 1987.

[33] C.C. Weems, S.P. Levitan, A.R. Hanson, E.M. Riseman, J.G. Nash, and D.B. Shu, “The Image Understanding Architecture,” *Int’l J. Computer Vision*, vol. 2, pp. 251–282, 1989.

[34] C.C. Weems and J.H. Burrill, “The Image Understanding Architecture and its Programming Environment,” *Parallel Architectures and Algorithm for Image Understanding*, V.K. Prasanna Kumar, ed. Academic Press, 1991.

[35] C.C. Weems, “The Performance of the Image Understanding Architecture on the DARPA Integrated Image Understanding Benchmark,” *Proc. Frontiers of Massively Parallel Computation*, pp. 361–364, Oct. 1988.

Ju-wook Jang received the BS degree in electronic engineering from Seoul National University in 1983, the MS degree in electrical engineering from the Korea Advanced Institute of Science and Technology in 1985, and the PhD degree in electrical engineering from the University of Southern California in 1993. Since 1995, he has been on the faculty of the Department of Electronic Engineering, Sogang University. From 1985 to 1988 and from 1993 to 1995, Dr. Jang worked with Samsung Electronics in the field of communication and computer development. His research interests include parallel architectures, parallel algorithms, and multimedia.

Heonchul Park received the PhD in computer engineering from the University of Southern California, Los Angeles, California, in 1993. He received his MS and BS degrees from Seoul National University, Seoul, Korea, in 1987 and 1985, respectively. Dr. Park has been engaged in microprocessor architecture designs for media signal processing since he joined the Micro-Electronics Division, Samsung Electronics Co., Ltd., Korea, in 1993. His interests include computer architecture, parallel processing, computational aspects on VLSI, and digital signal processing. He has had more than 30 technical papers published in international journals and conferences.

Viktor K. Prasanna (V.K. Prasanna Kumar) received his BS in electronics engineering from the Bangalore University and his MS from the School of Automation, the Indian Institute of Science. He obtained his PhD in computer science from the Pennsylvania State University in 1983. Currently, he is a professor in the Department of Electrical Engineering-Systems, University of Southern California, Los Angeles, and serves as director of the computer engineering division. His research interests include parallel computation, computer architecture, VLSI computations, and high performance computing for signal and image processing and vision.

Dr. Prasanna has published extensively and has served as a consultant for industries in the above areas. He has served on the organizing committees of several international meetings in VLSI computations, parallel computation, and high performance computing. He is the general co-chair of the IEEE International Parallel Processing Symposium, 1997, and has been the key person in developing the meeting into a premier international meeting in parallel computing. He also serves on the editorial boards of the *Journal of Parallel and Distributed Computing* and *IEEE Transactions on Computers*. He was the founding chair of the IEEE Computer Society Technical Committee on Parallel Processing. He is a fellow of the IEEE.