given k and n, one can choose any value of m in the range from 2 to n to construct a multiway merging network. However, use of certain values of m results in less delay than others for the same value of n. One choice of the value of m is $m = n$. In this case, $T(k,n,n) = T(k) + \lceil \log_2 n \rceil \lceil \log_2 k \rceil$. Other values of m, which can achieve this time bound for the given value of n, are $m = 2^c$ with $c < \lceil \log_2 n \rceil$.

We have also assumed that there are n keys in each ascending sequence and m divides n for simplicity and clarity of exposition. In general, a multiway merging network can be constructed to merge k sorted lists of different lengths with the ith list having n_i keys not necessarily equal to n. One way of doing this is to construct a k-way merger of size n first using keys with the value of positive infinity to fill the difference, and then remove these positive infinity keys together with the comparison-exchange elements associated with them, resulting in a simpler k-way merging network. Thus, there are no restrictions on the property of n_i, k, and m.

A sorter can be constructed from k-way mergers: the keys are combined k at a time to form ordered lists of length k; these lists are merged k at a time to form ordered lists of length k^2, etc., until all keys are merged into one ordered list. To sort k^p keys using the mergers requires k^{p-1} mergers of size 1 followed by k^{p-2} mergers of size k followed by k^{p-3} mergers of size k^2 followed by k^{p-4} mergers of size k^3, etc., etc. The longest path will go through $\sum_{i=0}^{p-1} T(k,m,k^i) = pT(k) + \lceil \log_2 k \rceil \lceil \log_2 m \rceil (\sum_{i=1}^{p-1} \lceil i \log_m k \rceil)$ steps.

V. Conclusion

The multiway merge described in this paper merges k ascending sequences into an ascending sequence for any integer k. This differs from existing merging networks that merge only two ascending sequences into one. Furthermore, the k-way merge represents a complete generalization of the odd-even merge, when $k = 2$. In this case, it uses m small two-way mergers to merge two ascending sequences into one, where m is not restricted to 2. The odd-even merge is a special case of the k-way merge, when $k = 2$ and $m = 2$. The multiway merges does not reduce the number of comparators nor does it reduce the delay over the odd-even merge. However, it does provide a comprehensive extension and generalization of the odd-even merge method, allowing more flexible construction of merge sorting networks.

References

[1] K. E. Batcher, "Sorting networks and their applications," in *AFIPS Proc. Spring Joint Comput. Conf.*, 1968, pp. 307–314.

[2] K. E. Batcher, "On bitonic sorting networks," in *Proc. 19th Int. Conf. Parallel Process.*, 1990, pp. 376–379.

[3] D. E. Knuth, "Sorting and searching," *The Art of Computer Programming*. Reading, PA: Addison-Wesley, 1973, vol. 3.

Design of Efficient Regular Arrays for Matrix Multiplication by Two-Step Regularization

Jong-Chuang Tsay and Pen-Yuang Chang

***Abstract*—A two-step regularization method in which first permutation sequences and then broadcast planes are selected is proposed to design various regular iterative algorithms for matrix multiplication. The regular iterative algorithms are then spacetime mapped to regular arrays, such as mesh, cylindrical, two-layered mesh, and orbital arrays. The proposed method can be used to design regular arrays with execution time of less than N (problem size).**

***Index Terms*— Broadcast, cylindrical array, mesh array, orbital array, parallel algorithm design, permutation sequence, propagation, two-layered mesh array, VLSI architecture**

I. Introduction

The topic of designing 2-D regular arrays for matrix multiplication has been studied for over a decade. Most existing designs are based on the well-known sequential algorithm of $C = A \times B$ (A, B, C are all $N \times N$ matrices). These include, in chronological order, the hexagonal array (with execution time on the order of $5N$) proposed by Kung and Leiserson [1], the mesh array ($3N$) proposed by Kung [2], the hexagonal array ($3N$) of Li and Wah [3], the orbital array (N) of Porter and Aravena [4], the cylindrical array ($2N$) of Porter and Aravena [5], the two-layered mesh array ($2N$) of Kak [6], and the cylindrical array, two-layered mesh array, and mesh array ($\frac{3N}{2}$) proposed by Tsay and Chang [7]. Other designs are based on the *Winograd* algorithm. These include the mesh array ($\frac{5N}{2}$) proposed by Jagadish and Kailath [8] and the two-layered mesh array ($\frac{3N}{2}$) presented by Benaini and Robert [9].

Some of the regular array designs based on $C = A \times B$ were proposed in a rather ad hoc fashion. Although they are derived from the same sequential algorithm, no one has ever written regular iterative algorithms (RIA's) [10] for all of them and stated the relationship between them. To derive these RIA's in a unified way, in this paper we propose a *two-step regularization* method: in the first step a permutation sequence is selected for each index and in the second step a broadcast plane is selected for each variable. Then, by spacetime mapping, various regular arrays can be designed. Furthermore, with knowledge of the derivation of these RIA's, regular arrays with execution time of less than N, faster than any other designs we know of, can be obtained.

This paper is organized as follows: Section II presents some preliminary definitions. In Section III, the two-step regularization method is proposed. Using this method, we can design RIA's in a unified manner for mesh arrays, cylindrical arrays, and two-layered mesh arrays. In Section IV, we design regular arrays with execution time approaching and then equal to N. Section V studies orbital array derivations with execution time of less than N. Finally, concluding remarks are presented in Section VI.

Manuscript received February 18, 1992; revised April 1, 1994. This work was supported by the National Science Council of the R.O.C. under Contract NSC-81-0408-E-009-568.

The authors are with Institute of Computer Science and Information Engineering, College of Engineering, National Chiao Tung University, Hsinchu, Taiwan 30050, R.O.C.

IEEE Log Number 9406340.

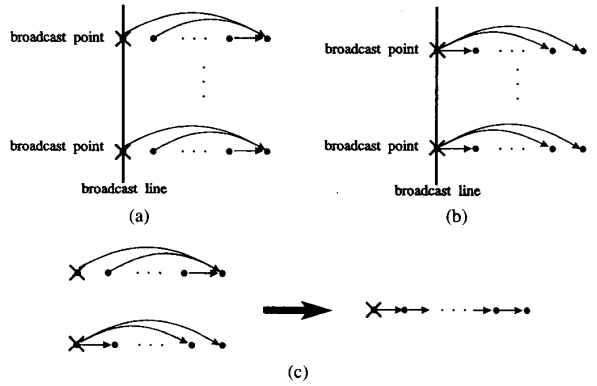

Fig. 1. (a) Multiple fan-in, broadcast point, and broadcast line. (b) Multiple fan-out, broadcast point, and broadcast line. (c) Transformation of broadcast vectors to propagation vectors.

II. Preliminary Definitions

In this section, we give some preliminary definitions as a basis for the descriptions that follow. The permutation sequences introduced here are useful for determining which data will appear in which position and for constructing the various types of links between nodes in a dependence graph (DG) [10]—a graphical representation of an RIA.

Definition 2.1: A *permutation sequence* $P(1,N) = (p(1), p(2),\cdots,p(N))$ is a sequence of integers resulting from permuting the sequence $(1,2,\cdots,N)$. The ith element in P is denoted by $p(i)$, where $1 \le i \le N$.

By observing the types of links in various regular arrays, we define four types of permutation sequences. They are the increasing sequence, for horizontal (or vertical) links, the left-shift sequence, for spiral links, and the even-odd and odd-even transposition sequences, for diagonal links.

Definition 2.2: A permutation sequence $U(1,N) = (u(1), u(2),\cdots,u(N))$ is an *increasing sequence* if the ith element in U is $u(i) = i$, where $1 \le i \le N$.

Definition 2.3: A permutation sequence $L(i;1,N) = (l_i(1), l_i(2),\cdots,l_i(N))$ is a left-shift sequence if it is formed from shifting the sequence $(1,2,\cdots,N)$ left cyclically i times, where $0 \le i \le N-1$. The jth element in L is $l_i(j) = (i+j-1)_{\mathrm{mod}N}+1$, where $1 \le j \le N$.

Definition 2.4: A permutation sequence $E(i;1,N) = (e_i(1), e_i(2),\cdots,e_i(N))$ is an even-odd transposition sequence if the jth element in E is

$$e_i(j) = \begin{cases} j-i, & \text{if } i+j \text{ is even } \wedge\, j > i \\ i-j+1, & \text{if } i+j \text{ is even } \wedge\, j \le i \\ i+j, & \text{if } i+j \text{ is odd } \wedge\, i+j \le N \\ 2N+1-i-j, & \text{if } i+j \text{ is odd } \wedge\, i+j > N \end{cases}$$

where $0 \le i \le N-1$ and $1 \le j \le N$.

Definition 2.5: A permutation sequence $O(i;1,N) = (o_i(1),o_i(2),\cdots,o_i(N))$ is an odd-even transposition sequence if the jth element in O is

$$o_i(j) = \begin{cases} i+j, & \text{if } i+j \text{ is even } \wedge\, i+j \le N \\ 2N+1-i-j, & \text{if } i+j \text{ is even } \wedge\, i+j > N \\ j-i, & \text{if } i+j \text{ is odd } \wedge\, j > i \\ i-j+1, & \text{if } i+j \text{ is odd } \wedge\, j \le i \end{cases}$$

where $0 \le i \le N-1$ and $1 \le j \le N$.

The multiple fan-in (Fig. 1(a)) and multiple fan-out (Fig. 1(b)) data dependence vectors are called *broadcast vectors*. All broadcast vectors can be systematically transformed into propagation vectors (e.g., Fig. 1(c)) [11]. We use the term *broadcast point* to denote the starting position for data propagation. A *broadcast line* is composed of several broadcast points. By aggregating broadcast lines, we obtain a *broadcast plane*.

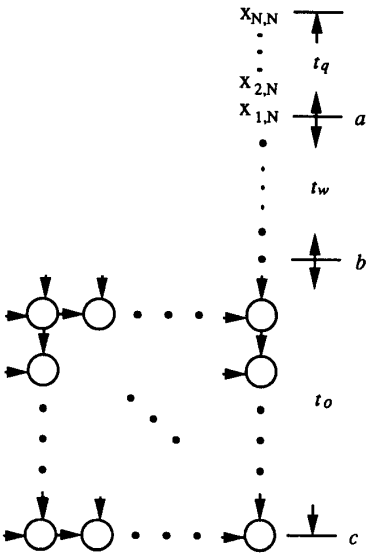

Fig. 2. The execution time $t_e = t_q + t_w + t_o$.

The execution time (t_e) of a regular array is defined as the time interval between the time when the first operation is executed and the time when the last result is calculated. t_e can be decomposed into three parts: the queuing time (t_q), the waiting time (t_w), and the operating time (t_o) (Fig. 2). t_q is the time interval between the time execution begins and the time when $x_{N,N}$ arrives at position a. t_w is the time interval between the time when $x_{N,N}$ leaves position a and the time when $x_{N,N}$ arrives at position b; this time is incurred because $x_{N,N}$ must wait to meet another datum at the first PE. t_o is the time interval between the time when $x_{N,N}$ leaves position b and the time when $x_{N,N}$ arrives at position c; in this time interval, $x_{N,N}$ actually operates with other data.

III. Two-Step Regularization

The matrix multiplication can be carried out in N recursions, as described in Algorithm 3.1.

Algorithm 3.1:
For $i = 1$ to N
For $j = 1$ to N
For $k = 1$ to N
$\quad c_{i,j} = c_{i,j} + a_{i,k} \times b_{k,j}$
initially $c_{i,j} = 0$.

Before the selection of a permutation sequence for each index, we must fully index all variables [12]. This can be done by substituting $a(i,j,k)$, $b(i,j,k)$, and $c(i,j,k)$ for $a_{i,k}$, $b_{k,j}$, and $c_{i,j}$, respectively, in Algorithm 3.1. Let $p^1(\alpha)$, $p^2(\beta)$, and $p^3(\gamma)$ denote the permutation sequences for indexes i, j, and k, respectively, where α, β, γ are functions of (i,j,k). Then Algorithm 3.1 can be rewritten as follows:

Algorithm 3.2:
For all indices (i,j,k), $1 \le i,j,k \le N$, do
$\quad c(i,j,k) = a(i,j,k) \times b(i,j,k)$
$\quad a(i,j,k) = a_{p^1(\alpha),p^3(\gamma)}$
$\quad b(i,j,k) = b_{p^3(\gamma),p^2(\beta)}$
Final results $c_{p^1(\alpha),p^2(\beta)} = c(i,j,1) + \cdots + c(i,j,N)$.

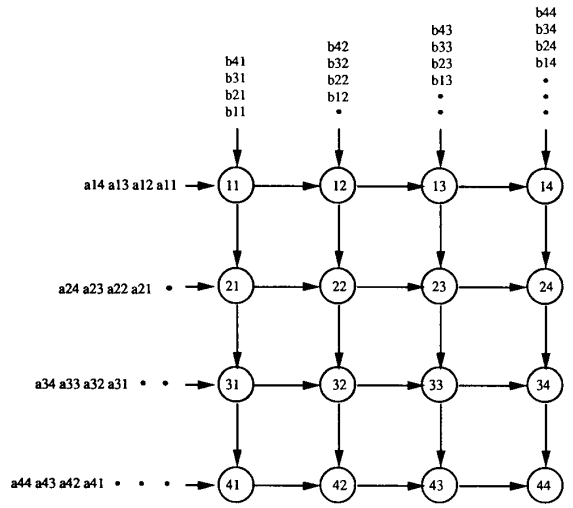

Fig. 3. Mesh array of Design $mm1$ ($N = 4$).

A. Step 1: Select Permutation Sequence for Each Index

A selection of $p^1(\alpha)$, $p^2(\beta)$, and $p^3(\gamma)$ is said to be *correct* if the final results of Algorithm 3.2 are correct for calculating matrix multiplication, i.e., if $c_{p^1(\alpha),p^2(\beta)} = a_{p^1(\alpha),1} \times b_{1,p^2(\beta)} + \cdots + a_{p^1(\alpha),N} \times b_{N,p^2(\beta)}$, $\forall\ p^1(\alpha)$, $p^2(\beta)$. Thus, the simplest way to select $p^3(\gamma)$ is to let $p^3(\gamma) = u(k) = k$. The selection of $p^1(\alpha)$ and $p^2(\beta)$ is based on the types of links (e.g., horizontal, vertical, spiral, or diagonal) given by the regular array that is to be designed. Horizontal or vertical links correspond to the increasing sequence; spiral links correspond to the left-shift sequence; and diagonal links correspond to the even-odd (or odd-even) transposition sequence.

For example, a mesh array has only horizontal and vertical links, so we select $u(i)$, $u(j)$, and $u(k)$ for $p^1(\alpha)$, $p^2(\beta)$, and $p^3(\gamma)$, respectively. Then we obtain Algorithm 3.3. It is easy to see that the selection is correct, because Algorithm 3.3 can correctly perform matrix multiplication.

Algorithm 3.3:

For all indices (i,j,k), $1 \le i,j,k \le N$, do
$c(i,j,k) = a(i,j,k) \times b(i,j,k)$
$a(i,j,k) = a_{i,k}$
$b(i,j,k) = b_{k,j}$
final results $c_{i,j} = c(i,j,1) + \cdots + c(i,j,N)$.

Another example is a cylindrical array, which has two types of links. For the first type, the horizontal links, we select the increasing sequence. For the second, the spiral links, we select the left-shift sequence. Thus, we select $u(i)$, $l_{i-1}(j)$, and $u(k)$ for $p^1(\alpha)$, $p^2(\beta)$, and $p^3(\gamma)$, respectively. The result is Algorithm 3.4.

Algorithm 3.4:

For all indices (i,j,k), $1 \le i,j,k \le N$, do
$c(i,j,k) = a(i,j,k) \times b(i,j,k)$
$a(i,j,k) = a_{i,k}$
$b(i,j,k) = b_{k,l_{i-1}(j)}$
final results $c_{i,l_{i-1}(j)} = c(i,j,1) + \cdots + c(i,j,N)$.

B. Step 2: Select Broadcast Plane for Each Variable

Various broadcast planes can be selected depending on the constraints of I/O bandwidth, I/O port location, supported hardware and so forth of a regular array. Then, broadcast vectors are transformed into propagation vectors by the method proposed in [11]. Finally, spacetime mapping is applied to derive regular arrays. From Algorithm 3.3, if the I/O ports of the regular array are confined at boundary PE's, then the broadcast planes $j = 1$, $i = 1$, and $k = 1$ are selected for variables a, b, and c, respectively. For the variable a, we have $a(i,j,k) = a(i,1,k)$, $\forall 2 \le j \le N$, and $a(i,1,k) = a_{i,k}$. It is easy to replace the broadcast vectors [0 l 0], $1 \le l \le N-1$, by the propagation vector [0 1 0]. Then we have the recurrence equation $a(i,j+1,k) = a(i,j,k)$. Applying the same method to variables b and c, we obtain Algorithm 3.5. The correctness of an RIA can be checked by resubstituting initial values into the recurrence equations and calculating the final results. That is, if $c_{i,j}$ is equal to $a_{i,1} \times b_{1,j} + \cdots + a_{i,N} \times b_{N,j}$, for $1 \le i,j \le N$, then the RIA is correct. It is easy to prove that Algorithm 3.5 is correct. A mesh array (Fig. 3) can be derived by projecting Algorithm 3.5 in the k-direction. This well-known mesh array was devised by Kung in [2]. We call this Design $mm1$. Its execution time is $t_e = t_q + t_w + t_o = (N-1) + (N-1) + (N) = 3N - 2$.

Algorithm 3.5:

For all indices (i,j,k), $1 \le i,j,k \le N$, do
$c(i,j,k+1) = c(i,j,k) + a(i,j,k) \times b(i,j,k)$
$a(i,j+1,k) = a(i,j,k)$
$b(i+1,j,k) = b(i,j,k)$
initial values
$c(i,j,1) = 0$
$a(i,1,k) = a_{i,k}$
$b(1,j,k) = b_{k,j}$
final results $c_{i,j} = c(i,j,N+1)$.

Again, with Algorithm 3.3, if the I/O ports of the regular array are located at the diagonal PE's, then the broadcast plane $i = j$ is selected for both variables a and b. Thus, we have broadcast vectors [0 l 0] for the variable a, with $1 \le l \le N-1$ if $i \le j$ and $1-N \le l \le -1$ if $i \ge j$, and [l 0 0] for the variable b, with $1 \le l \le N-1$ if $i \ge j$ and $1-N \le l \le -1$ if $i \le j$. In this way, we decompose the computation domain into two phases: $i \le j$ and $i \ge j$. In the $i \le j$ phase, the propagation vectors of variables a and b are in the [0 1 0] and [−1 0 0] directions, respectively. In the $i \ge j$ phase, the propagation vectors of variables a and b are in the [0 −1 0] and [1 0 0] directions, respectively. Finally, we obtain Algorithm 3.6. By projecting this algorithm in the k-direction, we obtain a mesh array (Fig. 4), which we call Design $mm2$. Its execution time is $t_e = t_q + t_w + t_o = (N-1) + 0 + (N) = 2N - 1$. An alternative way to calculate t_e is by the two-phase linear schedule proposed in [7].

Algorithm 3.6:

[*Phase 1*]: $i \le j$
For all indices (i,j,k), $1 \le i,k \le N$, $i \le j \le N$, do
$c(i,j,k+1) = c(i,j,k) + a(i,j,k) \times b(i,j,k)$
$a(i,j+1,k) = a(i,j,k)$
$b(i-1,j,k) = b(i,j,k)$
[*Phase 2*]: $i \ge j$
For all indices (i,j,k), $1 \le i,k \le N$, $1 \le j \le i$, do
$c(i,j,k+1) = c(i,j,k) + a(i,j,k) \times b(i,j,k)$
$a(i,j-1,k) = a(i,j,k)$
$b(i+1,j,k) = b(i,j,k)$
initial values
$c(i,j,1) = 0$
$a(i,i,k) = a_{i,k}$
$b(j,j,k) = b_{k,j}$
final results $c_{i,j} = c(i,j,N+1)$.

For Algorithm 3.4, if the I/O ports are confined at the boundary PE's, then we select the broadcast plane $j = 1$ for both variables a and b and the $(k = 1)$-plane for the variable c. For the variable b, from $b(i,j,k) = b_{k,l_{i-1}(j)}$ in Algorithm 3.4 and the broadcast plane $j = 1$, we can derive recurrence equations $b(i-1,j+1,k) = b(i,j,k)$

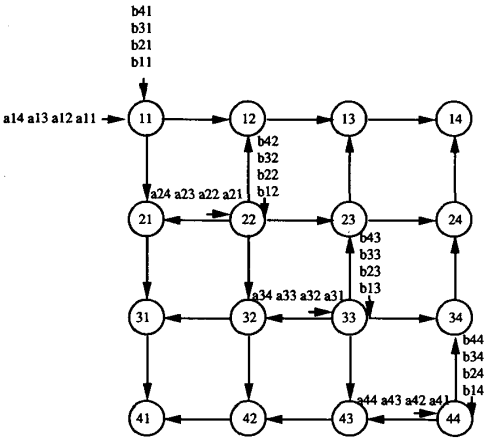

Fig. 4. Mesh array of Design $mm2$.

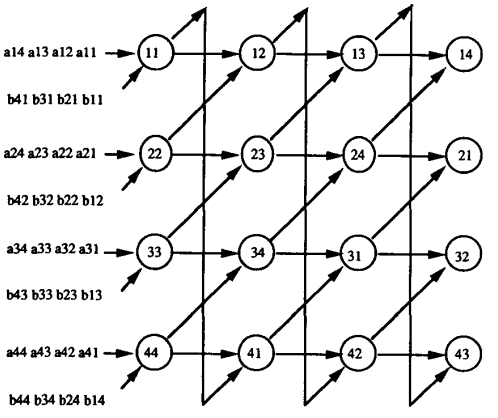

Fig. 5. Cylindrical array of Design $mm3$.

from $b(i-1,j+1,k) = b_{k,l_{i-2}(j+1)} = b_{k,l_{i-1}(j)} = b(i,j,k)$ and $b(N,j+1,k) = b(1,j,k)$ from $b(N,j+1,k) = b_{k,l_{N-1}(j+1)} = b_{k,l_0(j)} = b(1,j,k)$. Combining these recurrence equations with $b(i,1,k) = b_{k,l_{i-1}(1)}$, we have Algorithm 3.7. By projecting this algorithm in the k-direction, the cylindrical array proposed in [5] is obtained (Fig. 5). We call this Design $mm3$. Its execution time is $t_e = t_q + t_w + t_o = (N-1) + 0 + (N) = 2N-1$.

Algorithm 3.7:

For all indices (i,j,k), $1 \le i,j,k \le N$, do

$c(i,j,k+1) = c(i,j,k) + a(i,j,k) \times b(i,j,k)$

$a(i,j+1,k) = a(i,j,k)$

$b(i-1,j+1,k) = b(i,j,k)$ if $i \neq 1$

$b(N,j+1,k) = b(i,j,k)$ if $i = 1$

initial values

$c(i,j,1) = 0$

$a(i,1,k) = a_{i,k}$

$b(i,1,k) = b_{k,l_{i-1}(1)}$

final results $c_{i,l_{i-1}(j)} = c(i,j,N+1)$.

The two-layered mesh array proposed in [6] can be designed by letting $p^1(\alpha) = o_{i-1}(j)$, $p^2(\beta) = e_{i-1}(j)$ and $p^3(\gamma) = u(k)$ and choosing the broadcast plane $i = 1$ for variables a and b and the $(k = 1)$-plane for the variable c. It is not difficult to derive Algorithm 3.8. By projecting this algorithm in the k-direction, we obtain a two-layered mesh array (Fig. 6) with execution time

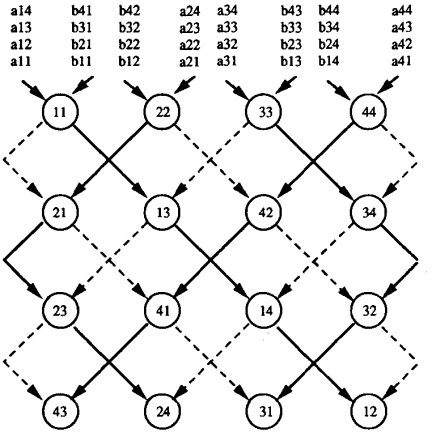

Fig. 6. Two-layered mesh array of Design $mm4$.

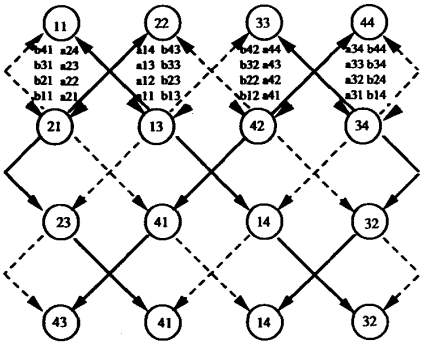

Fig. 7. Two-layered mesh array of Design $mm5$.

$t_e = 2N-1$. We call this Design $mm4$. Furthermore, if the same permutation sequences but the broadcast plane $i = \lceil \frac{N}{2} \rceil$ are selected for variables a and b, a two-layered mesh array with execution time $t_e = \lceil \frac{3N-1}{2} \rceil$ can be obtained (Fig. 7). We call this Design $mm5$. It has been proposed in [7].

Algorithm 3.8: For all indices (i,j,k), $1 \le i,j,k \le N$, do

$$c(i,j,k+1) = c(i,j,k) + a(i,j,k) \times b(i,j,k)$$

$$a(i,j,k) = \begin{cases} a(i-1,j-1,k), & \text{if } i+j \text{ is even } \wedge\, j \neq 1 \\ a(i-1,j+1,k), & \text{if } i+j \text{ is odd } \wedge\, j \neq N \\ a(i-1,j,k), & \text{if } i+j \text{ is even } \wedge\, j = 1 \\ a(i-1,j,k), & \text{if } i+j \text{ is odd } \wedge\, j = N \end{cases}$$

$$b(i,j,k) = \begin{cases} b(i-1,j-1,k), & \text{if } i+j \text{ is odd } \wedge\, j \neq 1 \\ b(i-1,j+1,k), & \text{if } i+j \text{ is even } \wedge\, j \neq N \\ b(i-1,j,k), & \text{if } i+j \text{ is odd } \wedge\, j = 1 \\ b(i-1,j,k), & \text{if } i+j \text{ is even } \wedge\, j = N \end{cases}$$

initial values

$$c(i,j,1) = 0$$

$$a(1,j,k) = a_{j,k}$$

$$b(1,j,k) = b_{k,j}$$

final results $c_{o_{i-1}(j),e_{i-1}(j)} = c(i,j,N+1)$.

IV. Arrays with Execution Time Approaching N

In the previous section, each variable has only one broadcast plane in the computation domain. If the given regular array has more I/O

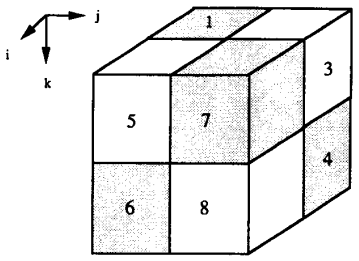

Fig. 8. The eight regions of the computation domain of matrix multiplication.

ports, then we can let a variable have more than one broadcast plane. That is, we let each variable have different broadcast planes in different regions of the computation domain. Then we can design regular arrays with execution time approaching or equal to N.

The computation domain of matrix multiplication can be divided into eight regions (Fig. 8). For simplicity, we assume N is an even number and define $i_1 \equiv 1 \le i \le \frac{N}{2}$, $i_2 \equiv \frac{N}{2}+1 \le i \le N$, $j_1 \equiv 1 \le j \le \frac{N}{2}$, $j_2 \equiv \frac{N}{2}+1 \le j \le N$, $k_1 \equiv 1 \le k \le \frac{N}{2}$, and $k_2 \equiv \frac{N}{2}+1 \le k \le N$. These eight regions are then denoted by $I_{111} \equiv i_1 \wedge j_1 \wedge k_1$ (region 1), $I_{112} \equiv i_1 \wedge j_1 \wedge k_2$ (region 2), $I_{121} \equiv i_1 \wedge j_2 \wedge k_1$ (region 3), $I_{122} \equiv i_1 \wedge j_2 \wedge k_2$ (region 4), $I_{211} \equiv i_2 \wedge j_1 \wedge k_1$ (region 5), $I_{212} \equiv i_2 \wedge j_1 \wedge k_2$ (region 6), $I_{221} \equiv i_2 \wedge j_2 \wedge k_1$ (region 7), and $I_{222} \equiv i_2 \wedge j_2 \wedge k_2$ (region 8). The broadcast plane in a region for a variable is denoted by, for example, I_{11_B1}, which implies $i_1 \wedge j_1 \wedge (j=1) \wedge k_1$, when the $(j=1)$-plane is selected for the broadcast plane in region 1.

For Algorithm 3.3, let the smallest possible values of i, j, and k be chosen as the broadcast planes for variables b, a, and c, respectively, in regions 1, 4, 6, and 7 (the shaded regions in Fig. 8). For example, $I_{11_B1} \equiv I_{111} \wedge (j=1)$ for the variable a in region 1, and $I_{2_B12} \equiv I_{212} \wedge (i = \frac{N}{2}+1)$ for the variable b in region 6. Then the variable a has broadcast planes I_{11_B1}, I_{12_B2}, I_{21_B2}, and I_{22_B1}, and the variable b has I_{1_B11}, I_{1_B22}, I_{2_B12}, and I_{2_B21}.

Since $a_{i,k}$ and $b_{k,j}$ are input in regions 1, 4, 6, and 7, the remaining regions obtain these input data from propagation vectors. That is, region 3 (and 8) obtains the input data $a_{i,k}$ of region 1 (and 6) from the propagation vector [0 1 0]. This vector can be carried out by the recurrence equation $a(i,j,k) = a(i,j-1,k)$. But region 2 (and 5) gets $a_{i,k}$ of region 4 (and 7) from the propagation vector [0 −1 0]. In regions 2 and 5, in order to ensure that every $a_{i,k}$ meets with its respective $b_{k,j}$, a new variable $a1$ is introduced to carry $a_{i,k}$. $a_{i,k}$ carried by $a1$ is moved leftward, reflected on $I_{11_B2} \vee I_{21_B1}$, and then moved rightward to meet with $b_{k,j}$. Hence, new recurrence equations, $a1(i,j,k) = a_{i,k}$ if $I_{12_B2} \vee I_{22_B1}$, $a1(i,j,k) = a1(i,j+1,k)$, and $a(i,j,k) = a1(i,j,k)$ if $I_{11_B2} \vee I_{21_B1}$, are added to propagate $a_{i.k}$ in regions 2 and 5. Similarly, for the variable b, we can derive recurrence equations $b1(i,j,k) = b_{k,j}$ if $I_{2_B12} \vee I_{2_B21}$, $b1(i,j,k) = b1(i+1,j,k)$, and $b(i,j,k) = b1(i,j,k)$ if $I_{1_B12} \vee I_{1_B21}$.

In regions 1, 2 (denoted by I_{11}) and regions 7, 8 (I_{22}), the variable c satisfies $c(i,j,k+1) = c(i,j,k) + a(i,j,k) \times b(i,j,k)$ with $c(i,j,1) = 0$. However, in regions 3, 4 (I_{12}) and regions 5, 6 (I_{21}), $c(i,j,\frac{N}{2}+1)$ is computed first, followed by the sequence $c(i,j,\frac{N}{2}+2), c(i,j,\frac{N}{2}+3), \cdots, c(i,j,N), c(i,j,1), \cdots, c(i,j,\frac{N}{2})$. To unify these two different recurrence equations for calculating the variable c in I_{11}, I_{22} and I_{12}, I_{21}, we introduce the binary operator $\oplus$, $i \oplus j \equiv (i+j-1)_{\mathrm{mod}N}+1$, $1 \le i,j \le N$. (Similarly, $\ominus$, $i \ominus j \equiv (i-j-1)_{\mathrm{mod}N}+1$, $1 \le i,j \le N$.) Then the recurrence equation for the variable c becomes $c(i,j,k \oplus 1) = c(i,j,k) + a(i,j,k) \times b(i,j,k)$, which is applied in all regions of the computation domain. Finally, we

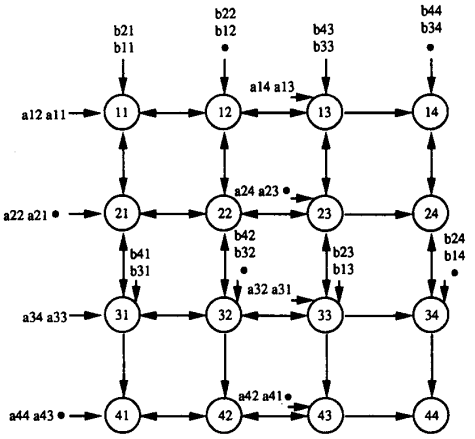

Fig. 9. Mesh array of Design $mm6$.

obtain Algorithm 4.1. A mesh array (Fig. 9) is obtained by projecting this algorithm in the k-direction. We call this Design $mm6$.

Algorithm 4.1: For all indices (i,j,k), $1 \le i,j,k \le N$, do

$$c(i,j,k \oplus 1) = c(i,j,k) + a(i,j,k) \times b(i,j,k)$$

$$a(i,j,k) = \begin{cases} a1(i,j,k) & \text{if } I_{11_B2} \vee I_{21_B1} \\ a(i,j-1,k) \end{cases}$$

$$a1(i,j,k) = a1(i,j+1,k)$$

$$b(i,j,k) = \begin{cases} b1(i,j,k) & \text{if } I_{1_B12} \vee I_{1_B21} \\ b(i-1,j,k) \end{cases}$$

$$b1(i,j,k) = b1(i+1,j,k)$$

initial values

$$c(i,j,1) = 0 \text{ if } I_{11} \vee I_{22}$$

$$c\left(i,j,\frac{N}{2}+1\right) = 0 \text{ if } I_{12} \vee I_{21}$$

$$a(i,j,k) = a_{i,k} \text{ if } I_{11_B1} \vee I_{12_B2} \vee I_{21_B2} \vee I_{22_B1}$$

$$b(i,j,k) = b_{k,j} \text{ if } I_{1_B11} \vee I_{1_B22} \vee I_{2_B12} \vee I_{2_B21}$$

$$a1(i,j,k) = a_{i,k} \text{ if } I_{12_B2} \vee I_{22_B1}$$

$$b1(i,j,k) = b_{k,j} \text{ if } I_{2_B12} \vee I_{2_B21}$$

final results

$$c_{i,j} = \begin{cases} c(i,j,1), & \text{if } I_{11} \vee I_{22} \\ c(i,j,\frac{N}{2}+1), & \text{if } I_{12} \vee I_{21} \end{cases}.$$

The execution time of Design $mm6$ is $t_e = t_q + t_w + t_o = (\frac{N}{2}-1) + (\frac{N}{2}-1) + N = 2N-2$. Comparing Design $mm6$ (Fig. 9) with its counterpart Design $mm1$ (Fig. 3), we see that the execution time has been reduced from $3N-2$ to $2N-2$, although the I/O bandwidth has been doubled.

The same design criteria can be applied to the other designs in the last section. For example, we apply this idea to Design $mm5$. If the I/O bandwidth is $2N$ $(4N, 8N, 16N, \cdots)$, then the execution time becomes $t_e \approx \frac{3N}{2}$ $(\frac{5N}{4}, \frac{9N}{8}, \frac{17N}{16}, \cdots)$. The extreme case is that where the I/O bandwidth becomes N^2 (or a and b reside in each PE initially), in which case the execution time is N. We call this Design $mm7$ (Fig. 10(a)). The RIA for design $mm7$ is depicted in Algorithm 4.2.

Algorithm 4.2: For all indices (i,j,k), $1 \le i,j,k \le N$, do the calculations on the bottom of the next page.

Notice that in this algorithm the value of an input variable is assigned as zero if its index is located outside the computation

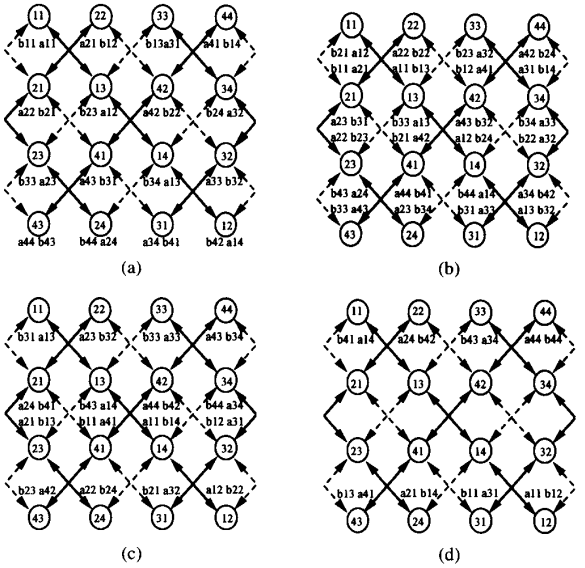

Fig. 10. Two-layered mesh array of Design $mm7$ and its snapshots.

domain. Snapshots of Design $mm7$ are shown in Fig. 10(a)–(d). For example, $c_{4,2}$ is computed at $PE_{2,3}$. First, $a_{4,2}$ and $b_{2,2}$ meet when $t = 1$; then when $t = 2$, $a_{4,3}$, $b_{3,2}$ and $a_{4,1}$, $b_{1,2}$ arrive at $PE_{2,3}$ simultaneously. Note that two copies of the multiplier and accumulator are necessary in that PE. Then, $a_{4,4}$ and $b_{4,2}$ arrive at $PE_{2,3}$ when $t = 3$. Finally, the results in these two accumulators are added to obtain the final result of $c_{4,2}$. Comparing Design $mm7$ and the orbital array in [4], we see that although both of them have execution time N, the former has the advantage that spiral links are not necessary.

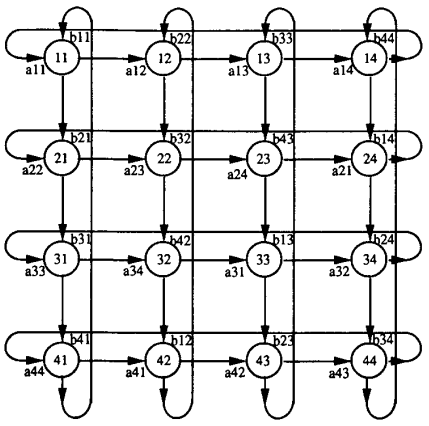

Fig. 11. Orbital array of Design $mm8$.

Porter and Aravena proposed an orbital array [4], yet they designed it in an ad hoc fashion. Designing an algorithm for an orbital array can be done by selecting the permutation sequences $p^1(\alpha) = u(i)$, $p^2(\beta) = u(j)$, and $p^3(\gamma) = u(k)$ and selecting the broadcast point $k = l_{i-1}(j)$ for all variables, so that we have $a(i,j,k) = a_{i,k}$, $b(i,j,k) = b_{k,j}$, and $c(i,j,k) = 0$ if $k = l_{i-1}(j)$. The orbital links are constructed by using the $\oplus$ and $\ominus$ operators. Thus we obtain Algorithm 4.3. By projecting this algorithm in the k-direction, we produce an orbital array (Fig. 11). We call this Design $mm8$. Its execution time is $t_e = t_q + t_w + t_o = 0 + 0 + N = N$.

Algorithm 4.3: For all indices (i,j,k), $1 \le i,j,k \le N$, do

$$c(i,j,k \ominus 1) = c(i,j,k) + a(i,j,k) \times b(i,j,k)$$
$$a(i,j \oplus 1,k) = a(i,j,k)$$
$$b(i \oplus 1,j,k) = b(i,j,k)$$

$$c1(i,j,k-1) = c1(i,j,k) + a1(i,j,k) \times b1(i,j,k)$$

$$a1(i,j,k) = \left.\begin{cases} a1(i-1,j-1,k), & \text{if } i+j \text{ is even } \wedge\, j \neq 1 \\ a1(i-1,j+1,k), & \text{if } i+j \text{ is odd } \wedge\, j \neq N \\ a1(i-1,j,k), & \text{if } i+j \text{ is even } \wedge\, j = 1 \\ a1(i-1,j,k), & \text{if } i+j \text{ is odd } \wedge\, j = N \end{cases}\right\} \wedge\, k \neq i$$

$$b1(i,j,k) = \left.\begin{cases} b1(i-1,j-1,k), & \text{if } i+j \text{ is odd } \wedge\, j \neq 1 \\ b1(i-1,j+1,k), & \text{if } i+j \text{ is even } \wedge\, j \neq N \\ b1(i-1,j,k), & \text{if } i+j \text{ is odd } \wedge\, j = 1 \\ b1(i-1,j,k), & \text{if } i+j \text{ is even } \wedge\, j = N \end{cases}\right\}$$

$$c2(i,j,k+1) = c2(i,j,k) + a2(i,j,k) \times b2(i,j,k)$$

$$a2(i,j,k) = \left.\begin{cases} a2(i+1,j+1,k), & \text{if } i+j \text{ is even } \wedge\, j \neq N \\ a2(i+1,j-1,k), & \text{if } i+j \text{ is odd } \wedge\, j \neq 1 \\ a2(i+1,j,k), & \text{if } i+j \text{ is even } \wedge\, j = N \\ a2(i+1,j,k), & \text{if } i+j \text{ is odd } \wedge\, j = 1 \end{cases}\right\} \wedge\, k \neq i$$

$$b2(i,j,k) = \left.\begin{cases} b2(i+1,j+1,k), & \text{if } i+j \text{ is odd } \wedge\, j \neq N \\ b2(i+1,j-1,k), & \text{if } i+j \text{ is even } \wedge\, j \neq 1 \\ b2(i+1,j,k), & \text{if } i+j \text{ is odd } \wedge\, j = N \\ b2(i+1,j,k), & \text{if } i+j \text{ is even } \wedge\, j = 1 \end{cases}\right\}$$

initial values

$$\left.\begin{aligned} &c1(i,j,k) = c2(i,j,k+1) = 0 \\ &a1(i,j,k) = a2(i,j,k) = a_{o_{i-1}(j),k} \\ &b1(i,j,k) = b2(i,j,k) = b_{k,e_{i-1}(j)} \end{aligned}\right\} \text{if } k = i$$

final results $c_{o_{i-1}(j),e_{i-1}(j)} = c1(i,j,0) + c2(i,j,N+1)$.

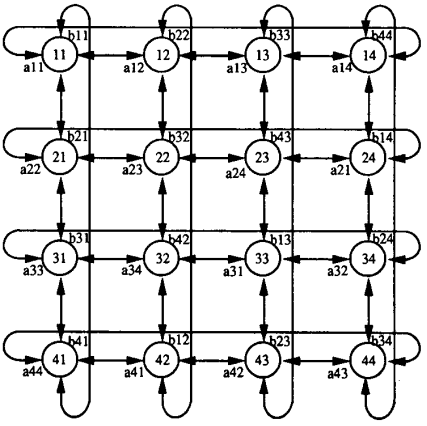

Fig. 12. Bidirectional orbital array of Design $mm9$.

initial values

$$\left.\begin{aligned} c(i,j,k) &= 0 \\ a(i,j,k) &= a_{i,k} \\ b(i,j,k) &= b_{k,j} \end{aligned}\right\} \text{ if } k = l_{i-1}(j)$$

final results $c_{i,j} = c(i,j,l_{i-1}(j))$.

V. Arrays with Execution Time Less Than N

To the best of our knowledge, N is the minimal execution time achieved to date for calculating matrix multiplication on a 2-D regular array. In this section, two orbital array derivations with execution time of $\frac{N}{2}$ are proposed.

The first array is obtained by expanding the uni-directional orbital array into a bidirectional one (Fig. 12). We call this Design $mm9$. In this design, two copies of the multiplier and accumulator are necessary in each PE. The RIA for Design $mm9$ is depicted in Algorithm 5.1. The execution time of Design $mm9$ is $t_e = t_q + t_w + t_o = 0 + 0 + \lceil \frac{N+1}{2} \rceil + 1 = \lceil \frac{N+1}{2} \rceil + 1$. The extra time step is for adding the results in the two accumulators of each PE.

Algorithm 5.1: For all indices (i,j,k), $1 \le i,j,k \le N$, do

$$\begin{aligned} c1(i,j,k \ominus 1) &= c1(i,j,k) + a1(i,j,k) \times b1(i,j,k) \\ a1(i,j \oplus 1,k) &= a1(i,j,k) \\ b1(i \oplus 1,j,k) &= b1(i,j,k) \\ c2(i,j,k \oplus 1) &= c2(i,j,k) + a2(i,j,k) \times b2(i,j,k) \\ a2(i,j \ominus 1,k) &= a(i,j,k) \\ b2(i \ominus 1,j,k) &= b(i,j,k) \end{aligned}$$

initial values

$$\left.\begin{aligned} c1(i,j,k) = c2(i,j,k \oplus 1) &= 0 \\ a1(i,j,k) = a2(i,j,k) &= a_{i,k} \\ b1(i,j,k) = b2(i,j,k) &= b_{k,j} \end{aligned}\right\} \text{ if } k = l_{i-1}(j)$$

final results $c_{i,j} = c1(i,j,l_{i-1}(j) \ominus \lceil \frac{N}{2} \rceil) + c2(i,j,l_{i-1}(j) \oplus \lceil \frac{N+1}{2} \rceil)$.

Yet another new design for a 2-D regular array with execution time of about $\frac{N}{2}$ is to let four pairs of input data a and b, $a_{i,k}$, $b_{k,j}$, $a_{i,k+\frac{N}{2}}$, $b_{k+\frac{N}{2},j}$, $a_{i+\frac{N}{2},k}$, $b_{k,j+\frac{N}{2}}$, and $a_{i+\frac{N}{2},k+\frac{N}{2}}$, $b_{k+\frac{N}{2},j+\frac{N}{2}}$ stay in each $\text{PE}_{i,j}$ on a unidirectional orbital array initially, where $1 \le i,j,k \le \frac{N}{2}$ and $k = l_{i-1}(j)$. The advantage of this arrangement is that two multiplications, $a_{i,k} \times b_{k,j}$ and $a_{i,k+\frac{N}{2}} \times b_{k+\frac{N}{2},j}$, in $\text{PE}_{i,j}$ for $c_{i,j}$ are computed at the same time. The same condition occurs on $c_{i,j+\frac{N}{2}}$, $c_{i+\frac{N}{2},j}$, and $c_{i+\frac{N}{2},j+\frac{N}{2}}$. Since computing each $c_{i,j}$ requires N multiplications of a and b and each time two of them can be done, the execution time is $t_e = \frac{N}{2} + 1$. The extra time step is for adding the results in the two accumulators of $\text{PE}_{i,j}$ for each $c_{i,j}$.

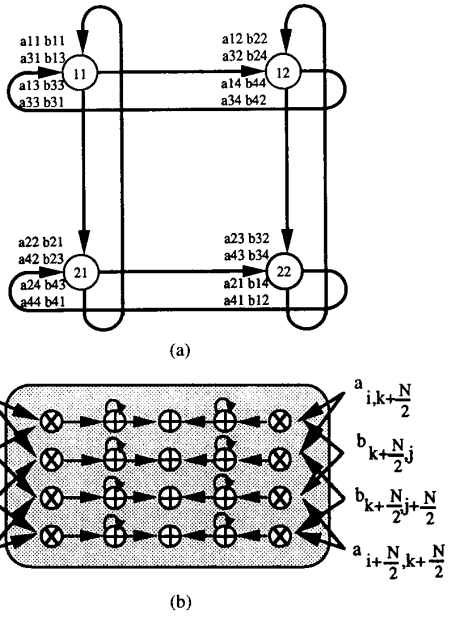

Fig. 13. (a) Orbital array of Design $mm10$. (b) The function of each PE.

From the DG viewpoint, the above design criteria are the first to cut the DG of Algorithm 4.3 into eight regions (Fig. 8), pile all of these eight regions into region 1, and finally project this new DG in the k-direction.

The RIA for this array can be derived by modifying Algorithm 4.3. First, the index (i,j,k) in Algorithm 4.3 is renamed (i',j',k'), and then the index (i',j',k') is expanded to $(\alpha,\beta,\gamma,i,j,k)$, where $i' = \alpha + i$, $j' = \beta + j$, $k' = \gamma + k$ and $i = (i'-1)_{\text{mod}\,\frac{N}{2}} + 1$, $j = (j'-1)_{\text{mod}\,\frac{N}{2}} + 1$, $k = (k'-1)_{\text{mod}\,\frac{N}{2}} + 1$. The new index $(\alpha,\beta,\gamma,i,j,k)$ is divided into two parts: (α,β,γ) and (i,j,k). The first part (α,β,γ) is used to indicate which one of the multipliers and accumulators will be used inside a PE; the second part (i,j,k) has the same effect of DG piling. In this way, we obtain Algorithm 5.2. Note that the binary operators $\oplus$ and $\ominus$ and the permutation sequence L are all defined for $\frac{N}{2}$ rather than N in this algorithm. Finally, by projecting the index (i,j,k) in Algorithm 5.2 in the k-direction, we obtain an orbital array with execution time $t_e = \frac{N}{2} + 1$ (Fig. 13(a)). The function of each PE is as shown in Fig. 13(b). We call this Design $mm10$.

Algorithm 5.2: For all indices $(\alpha,\beta,\gamma,i,j,k)$, $\alpha,\beta,\gamma \in \{0,\frac{N}{2}\}$, $1 \le i,j,k \le \frac{N}{2}$ do

$$\begin{aligned} c(\alpha,\beta,\gamma,i,j,k \ominus 1) &= c(\alpha,\beta,\gamma,i,j,k) + a(\alpha,\beta,\gamma,i,j,k) \\ &\quad \times b(\alpha,\beta,\gamma,i,j,k) \\ a(\alpha,\beta,\gamma,i,j \oplus 1,k) &= a(\alpha,\beta,\gamma,i,j,k) \\ b(\alpha,\beta,\gamma,i \oplus 1,j,k) &= b(\alpha,\beta,\gamma,i,j,k) \end{aligned}$$

initial values

$$\left.\begin{aligned} c(\alpha,\beta,\gamma,i,j,k) &= 0 \\ a(\alpha,\beta,\gamma,i,j,k) &= a_{i+\alpha,k+\gamma} \\ b(\alpha,\beta,\gamma,i,j,k) &= b_{k+\gamma,j+\beta} \end{aligned}\right\} \text{ if } k = l_{i-1}(j)$$

final results $c_{i',j'} = c(i'-i, j'-j, 0, i, j, l_{i-1}(j)) + c(i'-i, j'-j, \frac{N}{2}, i, j, l_{i-1}(j))$
where $i = (i'-1)_{\mathrm{mod}\,\frac{N}{2}} + 1$, $j = (j'-1)_{\mathrm{mod}\,\frac{N}{2}} + 1$.

VI. Conclusion

We have described a unified approach, two-step regularization, to derive the RIA's for matrix multiplication. The RIA's were then spacetime mapped to regular arrays. These regular arrays include mesh arrays, cylindrical arrays, two-layered mesh arrays, and orbital arrays. We note that the array type relies mainly on the permutation sequences and broadcast planes selected in the two-step regularization. The methodology proposed in this paper can be used to solve many other problems, especially problems that can be formulated in matrix form, e.g., LU-decomposition, transitive closure, and algebraic path problem. Using this methodology, we can design even faster regular arrays for these problems.

References

[1] H. T. Kung and C. E. Leiserson, "Systolic arrays for VLSI," in *Proc. 1978 Soc. Indust., Appl. Math.*, 1979, pp. 256–282.

[2] S. Y. Kung, "VLSI array processor for signal processing," in *Proc. Conf. Advanced Res. Integrat. Circuits*, 1980.

[3] G. J. Li and B. W. Wah, "The design of optimal systolic arrays," *IEEE Trans. Comput.*, vol. C-34, Jan. 1985, pp. 66–77.

[4] W. A. Porter and J. L. Aravena, "Orbital architectures with dynamic reconfiguration," in *IEE Proc.*, vol. 134, Nov. 1987, pp. 281–287.

[5] ——, "Cylindrical arrays for matrix multiplication," in *Proc. 24th Annu. Alleroton Conf. Commun., Control, Computing*, Mar. 1988, pp. 595–602.

[6] S. C. Kak, "A two-layered mesh array for matrix multiplication," *Parallel Computing*, vol. 6, pp. 383–385, 1988.

[7] J. C. Tsay and P. Y. Chang, "Some new designs of 2-D array for matrix multiplication and transitive closure," *IEEE Trans. Parallel, Distrib. Syst.*, to be published.

[8] H. V. Jagadish and T. Kailath, "A family of new efficient arrays for matrix multiplication," *IEEE Trans. Comput.*, vol. 38, pp. 149–155, Jan. 1989.

[9] A. Benaini and Y. Robert, "An even faster systolic array for matrix multiplication," *Parallel Computing*, vol. 12, pp. 249–254, 1989.

[10] S. K. Rao, "Regular iterative algorithms and their implementations on processor arrays," Ph.D. dissertation, Stanford Univ., Stanford, CA, 1985.

[11] Y. Wong and J. M. Delosme, "Transformation of broadcasts into propagations in systolic algorithms," *J. Parallel Distrib. Comput.*, vol. 14, pp. 121–145, 1992.

[12] S. Y. Kung, *VLSI Array Processor*. Englewood Cliffs, NJ: Prentice-Hall, 1988.