Unified approach to designing parallel Winograd algorithms

S. Yuan
J.-C. Tsay

Indexing terms: Cylindrical array, Matrix multiplication, Parallel algorithm

Abstract: Although the recurrence equation for the Winograd algorithm is uniform, no unified approach has been proposed to design parallel Winograd algorithms. In the paper the authors propose a unified approach to designing parallel Winograd algorithms. Using this approach, several parallel algorithms are designed. These algorithms are executed on regular arrays including conventional systolic arrays and nonplanar regular arrays. A comparison of their performance is given.

1 Introduction

There are many sequential algorithms for computing a matrix product, such as the standard multiplication algorithm [1–3], Winograd's algorithm [4, 5], and Strassen's algorithms [6]. Among these algorithms, the equations for standard multiplication algorithm, i.e.

$$c_{ij} = c_{ij} + \sum_{k=1}^{n} a_{ik} \times b_{kj} \quad \text{for } 1 \leqslant i, j \leqslant n$$

and the equations for the Winograd algorithm, i.e.

$$c_{ij} = \sum_{k=1}^{n/2} (a_{i,2k} + b_{2k-1,j}) \times (a_{i,2k-1} + b_{2k,j}) - \sum_{k=1}^{n/2} a_{i,2k} \times a_{i,2k-1} - \sum_{k=1}^{n/2} b_{2k,j} \times b_{2k-1,j}$$

$$\text{for } 1 \leqslant i, j \leqslant n$$

are suitable to be executed on regular arrays [7–13], because these equations are repeated and iterative. Based on the equations for standard multiplication algorithms, extensive researches on the design of parallel matrix multiplication algorithms have been carried out. These parallel algorithms include not only the algorithms for solving matrix multiplication problem but also for other matrix product-type problems such as band matrix multiplication [1, 14], bit-level matrix-vector multiplication problem, continuous matrix multiplication [15, 16], and discrete Fourier transformation [17]. However, only a few papers have used the equations for the Winograd algorithm to design parallel algorithms on regular arrays because its recurrence equation is less regular than that of the conventional standard multiplication algorithm.

Paper 9981E (C2), first received 22nd June and in revised form 20th September 1993

The authors are with the Institute of Computer Science and Information Engineering, College of Engineering, National Chiao Tung University, Hsinchu, Taiwan 30049, Republic of China

In Reference 4, it is said that an array architecture based on Winograd's algorithm cannot be obtained using a space–time mapping methodology [2], because neither the allocation function nor the timing function are quasi-affine. In this paper, we propose a unified approach to designing various parallel array architectures for the Winograd algorithm. The designs include both old and new algorithms; systolic algorithms and nonsystolic algorithms, such as those discussed in References 4 and 5.

In this paper we use the number of processors, total execution time, and the utilisation of each processor as criteria to compare the performance of various parallel Winograd algorithms. From this comparison, we conclude that the torus array algorithm have the shortest execution (excluding the loading and draining time) and the utilisation of each processor in the torus array algorithms is the highest.

2 Design methodology

Let n be even for the sake of simplicity. In the Winograd algorithm, the product $\boldsymbol{C} = \boldsymbol{A} \times \boldsymbol{B}$ is computed as

$$c_{ij} = d_{ij} - \alpha_i - \beta_j \tag{1}$$

where

$$d_{ij} = \sum_{k=1}^{n/2} (a_{i,2k} + b_{2k-1,j}) \times (a_{i,2k-1} + b_{2k,j})$$

$$\alpha_i = \sum_{k=1}^{n/2} a_{i,2k} \times a_{i,2k-1}$$

$$\beta_j = \sum_{k=1}^{n/2} b_{2k,j} \times b_{2k-1,j}$$

for $1 \leqslant i, j \leqslant n$.

The advantage of this algorithm is that the coefficient $\alpha_i(\beta_j)$ needs to be evaluated only once while it is used for the whole row i (column j) of the matrix $\boldsymbol{D}$. For convenience of analysis, we may rewrite the above equation as follows:

$$c_{ij} = \sum_{k=1}^{n/2} e_{ijk} \tag{2}$$

where

$$e_{ijk} = (a_{i,2k} + b_{2k-1,j}) \times (a_{i,2k-1} + b_{2k,j}) - \alpha_{ik} - \beta_{kj}$$

and

$$\alpha_{ik} = a_{i,2k} \times a_{i,2k-1}$$

$$\beta_{kj} = b_{2k,j} \times b_{2k-1,j}$$

for $1 \leqslant i, j \leqslant n$ and $1 \leqslant k \leqslant n/2$.

We see that one step of computation of eqn. 2 consists

of computing e_{ijk} in $\tau(e_{ijk})$ time units and adding e_{ijk} to c_{ij} in τ_{add} time units, so that the time unit τ in a synchronous array should be taken as $\tau = \tau(e_{ijk}) + \tau_{add}$. According to eqn. 2, we obtain the DG (dependence graph) of the Winograd algorithm shown in Fig. 1*a* (we use $n = 4$ as an example). In Fig. 1*a*, we see that data streams ***A***, ***B***, and ***C*** move in j, i and k-direction, respectively. Each node in the DG performs the computation of e_{ijk}.

In fact, $\alpha_{ik}(\beta_{kj})$ need be computed only once for the whole row i (column j) before computing e_{ijk}, therefore we can move the computation of α_{ik} and β_{kj} outside the DG of Fig. 1*a*. The results in a revised DG of Fig. 1*b*. The blank circular nodes in Fig. 1*b* perform the computation of α_{ik} or β_{kj}.

From Fig. 1*b*, we see that the computations of α_{ik} and β_{kj} play only a minor part of the whole computations, therefore, to simplify the description of various parallel Winograd algorithms, we will focus only on the time

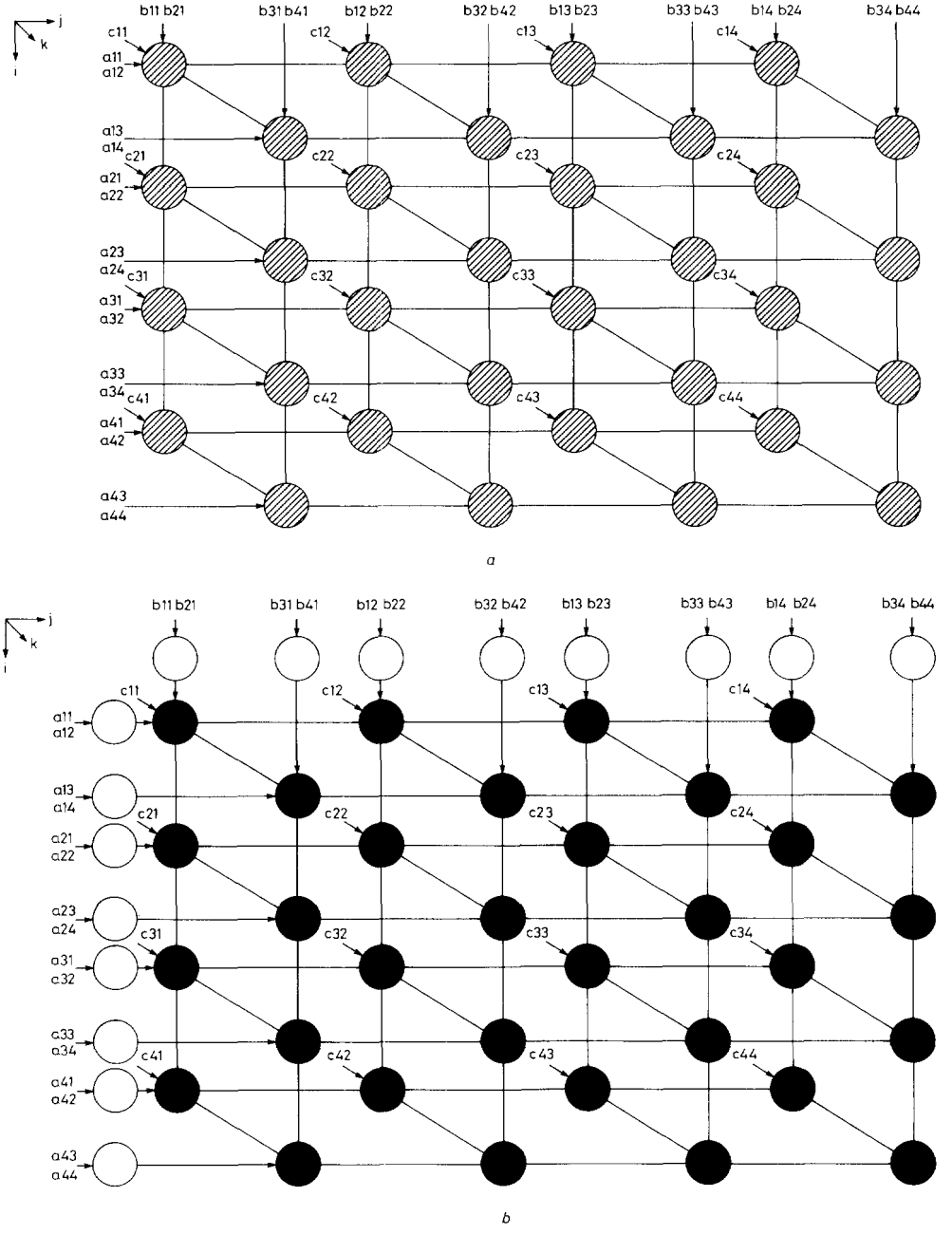


Fig. 1 *Winograd's algorithm*
a Dependence graph *b* Revised dependence graph

scheduling and processor assignment of the shaded circular nodes and ignore the time requirements for computing α_{ik} and β_{kj} on evaluating the total execution time of the parallel algorithm. To give a unified approach on the design of various types of parallel Winograd algorithms, we adopt the design method described in Reference 13. In Reference 13, the timing schedule and processor assignment of nodes in a DG are represented by a timing level table (TLT) [18] and a processor assignment table (PAT) [13], respectively. The TLT is a three-dimensional array and the PAT is a two-dimensional array. Let r, s, and q be the first, second, and third dimension of the TLT, respectively. Depending on the chosen projection, k, j or i-directions, $(r\ s\ q)$ is set to $(i\ j\ k)$, $(i\ k\ j)$ or $(k\ j\ i)$, respectively.

The number t_{rsq} on the position (r, s, q) of the TLT specifies that the computation of e_{ijk} is performed at time t_{rsq} and the number $p_{\gamma\delta}$ on the position (γ, δ) of the PAT specifies that the above computation is performed by the processor (γ, δ), where $p_{\gamma\delta} = (r, s)$. In other words, all the nodes $\{(r, s, q) | q = 1, 2, \ldots, n\}$ of the DG are projected (along the third dimension) onto the same processor (γ, δ). If we use [0 0 1] as the projection direction, then the processor index in the PAT is (i, j). If we use [1 0 0] as the projection direction, then the processor index in the PAT is (j, k).

Before introducing various designs for parallel algorithms, we first provide some definitions.

The utilisation U of processors in an algorithm is the average fraction of time that the processors are busy performing operations. Utilisation is computed as follows.

Let K be the number of processors,
T be the execution time, in units of τ, of the algorithm,
N be the number of primitive operations in the algorithm,
τ be the computation time of a primitive operation,

then

$$U = \frac{N\tau}{KT}$$

We use the following naming convention to specify various parallel algorithms. We divide the name into two parts. The first part specifies the type of algorithm and the second part specifies the selected projection direction. For the first part, we use ***S*** to denote a 'systolic' array algorithm, ***C*** a 'cylindrical' array algorithm [11], ***X*** a 'two-layered mesh' array algorithm [9], and ***MX*** a 'modified two-layered mesh' array algorithm. For the second part of the algorithm name, ***i***, ***j***, and ***k*** are used to denote that the selected projection direction are i, j and k-directions, respectively. Thus, algorithm ***Ck*** is a parallel algorithm obtained from projecting a DG along k-direction.

To adopt the design methodology of Reference 13, we need to construct a feasible TLT and then a PAT compatible with the TLT. Starting with the DG of Fig. 1*b* various parallel Winograd algorithms are designed as follows by constructing different pairs of the TLT and PAT.

2.1 Systolic array *Sk*

There have been many papers dealing with the design of conventional systolic arrays, so we omit it. A possible design instance ($n = 4$) of the TLT and PAT is shown in Table 1*a* and Table 1*b*. It corresponds to the parallel algorithm **Algorithm *Sk*** shown in Fig. 2 where circular processors are used to compute either α_{ik}'s or β_{kj}'s and rectangular processors are used to compute e_{ijk}. Total execution time of the algorithm is $(5n/2) - 2$ time steps, so the utilisation of each processor in **Algorithm *Sk*** is $n/2/(5n/2 - 2)$. This algorithm is implemented on a conventional systolic array. Execution sequence of this algorithm is shown in Table 2. From Table 2, we know that the utilisation of each processor is very low.

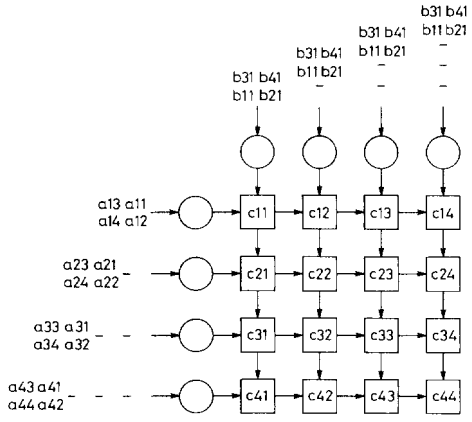


Fig. 2 *A systolic array for the Winograd algorithm*

2.2 Systolic array *Si*

To increase the utilisation of each processor, if we use i-direction as the projection direction and use Tables 3*a* and *b* as the TLT and PAT, then we obtain the parallel algorithm **Algorithm *Si*** shown in Fig. 3. This algorithm is also implemented on a conventional systolic array. Fig. 4 shows the operations performed by processors. Circular processors are used to compute α_{ik}'s and rectangular processors are used to compute both β_{kj}'s and e_{ijk}'s. Total execution time is also $5n/2 - 2$ time steps, so the utilisation of each processor is $n/(5n/2 - 2)$. This design is similar to the Winograd matrix multiplication array designed by Jagadish and Kailath [5].

Table 1: (*a*) TLT of algorithm *Sk*, (*b*) PAT of algorithm *Sk*

$k = 1$

$J =$	1	2	3	4
$i = 1$	1	2	3	4
2	2	3	4	5
3	3	4	5	6
4	4	5	6	7

$k = 2$

$j =$	1	2	3	4
$i = 1$	2	3	4	5
2	3	4	5	6
3	4	5	6	7
4	5	6	7	8

(*a*)

$\delta =$	1	2	3	4
$\gamma = 1$	11	12	13	14
2	21	22	23	24
3	31	32	33	34
4	41	42	43	44

(*b*)

Table 2: Execution sequence of Algorithm *Sk*

time step				
1	e_{111}	—	—	—
	—	—	—	—
	—	—	—	—
	—	—	—	—
2	e_{112}	e_{121}	—	—
	e_{211}	—	—	—
	—	—	—	—
	—	—	—	—
3	—	e_{122}	e_{131}	—
	e_{212}	e_{221}	—	—
	e_{311}	—	—	—
	—	—	—	—
4	—	—	e_{132}	e_{141}
	—	e_{222}	e_{231}	—
	e_{312}	e_{321}	—	—
	e_{411}	—	—	—
5	—	—	—	e_{141}
	—	—	e_{232}	e_{241}
	—	e_{322}	e_{331}	—
	e_{412}	e_{421}	—	—
6	—	—	—	—
	—	—	—	e_{242}
	—	—	e_{332}	e_{341}
	—	e_{422}	e_{431}	—
7	—	—	—	—
	—	—	—	—
	—	—	—	e_{342}
	—	—	e_{432}	e_{441}
8	—	—	—	—
	—	—	—	—
	—	—	—	—
	—	—	—	e_{442}

Table 3: (*a*) TLT of algorithm *Si*, (*b*) PAT of algorithm *Si*

$k=$	1	2
$j=1$	1	2
2	2	3
3	3	4
4	4	5

$i=1$

$k=$	1	2
$j=1$	2	3
2	3	4
3	4	5
4	5	6

$i=2$

$k=$	1	2
$j=1$	3	4
2	4	5
3	5	6
4	6	7

$i=3$

$k=$	1	2
$j=1$	4	5
2	5	6
3	6	7
4	7	8

$i=4$

(*a*)

$\delta=$	1	2
$\gamma=1$	11	21
2	12	22
3	13	23
4	14	24

(*b*)

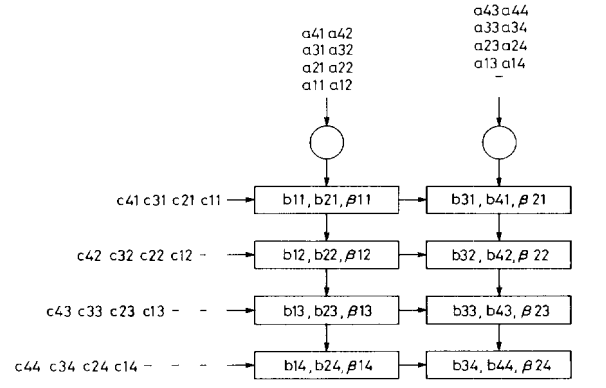


Fig. 3 *Another systolic array for the Winograd algorithm*

2.3 Cylindrical array ***Ck***

Now, we show how to design a cylindrical array for the Winograd algorithm. Assuming that the k-direction is selected as the projection direction, a feasible TLT $t = [t_{ijk}]$ is constructed by the following steps:

(i) Let $[t_{ij1}]$ be an ordered or permuted Latin square [13, 19].

(ii) Let $[t_{ijk}] = [t_{ij1} + (k-1)]$ for $k = 1, 2, \ldots, n/2$.

Then, we find a PAT compatible with the TLT we have just constructed. After determining the TLT and PAT, we can obtain a parallel algorithm. A possible design instance of the TLT and PAT is shown in Tables 4*a* and *b*. It corresponds to the parallel algorithm **Algorithm *Ck*** shown in Fig. 5. This algorithm is implemented on a cylindrical array. The total execution time is now reduced to $3n/2 - 1$ time steps. The utilisation of each processor is $n/2\,(3n/2 - 1)$.

2.4 Two-layered mesh array ***Xk***

If we use Table 4*a* and Table 5 as the TLT and PAT, then we obtain the parallel algorithm **Algorithm *Xk*** shown in Fig. 6. Total execution time and the utilisation of each processor is the same as **Algorithm *Ck***, but this architecture uses local connections instead of global con-

nections. The execution sequence of this algorithm is shown in Table 6.

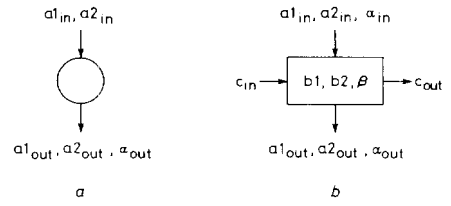


Fig. 4 *Processor*

a For computing α's
$a1_{out} := a1_{in}$
$a2_{out} := a2_{in}$
$\alpha_{out} := a1_{in} * a2_{in}$
b For computing β's and e_{ijk}'s
$\beta := b1 * b2$
β is assigned once only when first input data $(a1_{in}, a2_{in})$ received
$a1_{out} := a1_{in}$
$a2_{out} := a2_{in}$
$\alpha_{out} := \alpha_{in}$
$c_{out} := c_{in} + (a1_{in} + b2)(a2_{in} + b1) - \alpha_{in} - \beta$

links to drain out c_{ij} of $\boldsymbol{C}$ from the array. This algorithm is the same algorithm as that is proposed by Benaini and Robert [4]. Execution sequence of this algorithm is shown in Table 7. Comparing Table 6 with Table 7, we see that the TLT of **Algorithm *MXk*** is the same as that of **Algorithm *Xk***, but the utilisation of each processor for **Algorithm *MXk*** is $n/(3n/2 - 1)$, which is twice as much that is achieved by **Algorithm *Xk***.

*2.6 Cylindrical array **Ci***

If we use Tables 8*a* and *b* as the TLT and PAT, then we obtain the parallel algorithm **Algorithm *Ci*** shown in Fig. 8. The number of time steps required for this algorithm is $2n - 1$. The utilisation of each processor is $n/(2n - 1)$.

*2.7 Torus array **Tk***

If we use Tables 9*a* and *b* as the TLT and PAT, then we obtain the parallel algorithm **Algorithm *Tk*** shown in Fig. 9. The steps of designing a torus array algorithm is shown in the following:

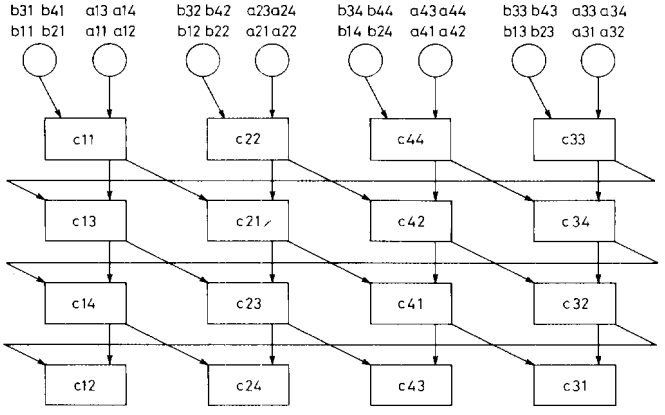


Fig. 5 *Cylindrical array for the Winograd algorithm*

Table 4: (*a*) TLT of algorithm *Ck*, (*b*) PAT of algorithm *Ck*

$j=$ / $i=$	1	2	3	4
1	1	4	2	3
2	2	1	3	4
3	4	3	1	2
4	3	2	4	1

$k = 1$

(*a*)

$j=$ / $i=$	1	2	3	4
1	2	5	3	4
2	3	2	4	5
3	5	4	2	3
4	4	3	5	2

$k = 2$

$\delta=$ / $\gamma=$	1	2	3	4
1	11	22	44	33
2	13	21	42	34
3	14	23	41	32
4	12	24	43	31

(*b*)

Table 5: PAT of algorithm *Xk*

$\delta=$ / $\gamma=$	1	2	3	4
1	11	22	33	44
2	21	13	42	34
3	23	41	14	32
4	43	24	31	12

*2.5 Modified two-layered mesh array **MXk***

Because the array of Fig. 6 is symmetrical to the central horizontal line, we can use the cut-and-pile method [20] by the central horizontal line to obtain the algorithm **Algorithm *MXk*** shown in Fig. 7, where we add vertical

(i) Find a TLT $t = [t_{rsq}]$ where $[t_{rs1}]$ is an ordered or a permuted Latin square and t is a Latin cube [19].

(ii) According to the data flow dependence graph, we can find a PAT compatible with the above TLT t.

After deciding the TLT and PAT, we obtain a torus array algorithm for the parallel Winograd algorithm. The number of time steps required for this algorithm is n. The utilisation of each processor is $\frac{1}{2}$.

*2.8 Torus array **Ti***

If we use Tables 10*a* and *b* as the TLT and PAT, then we obtain the parallel algorithm **Algorithm *Ti*** shown in Fig. 10. The number of time steps required for this algorithm is n. The utilisation of each processor is 1.

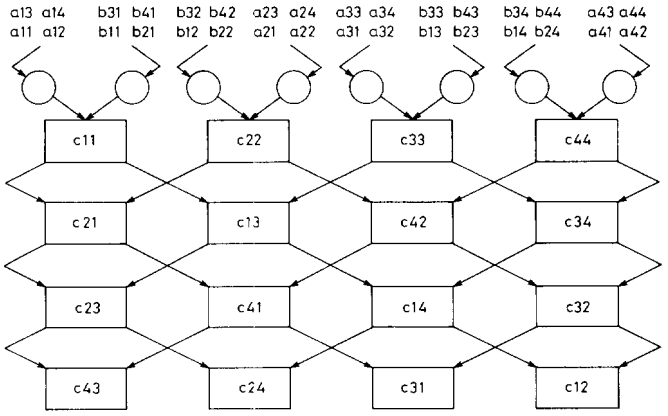


Fig. 6 *Two-layered mesh array for the Winograd algorithm*

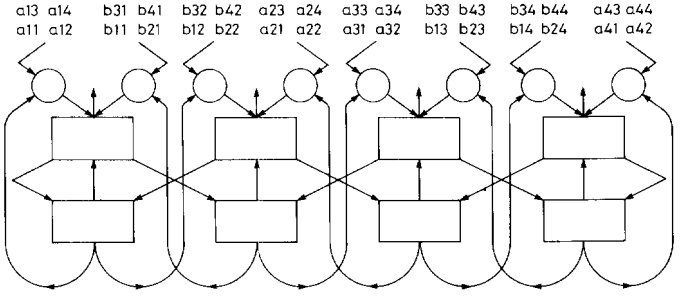


Fig. 7 *Modified two-layered mesh array for the Winograd algorithm*

Table 6: Execution sequence of algorithm *Xk*

time step 1

e_{111}	e_{221}	e_{441}	e_{331}
—	—	—	—
—	—	—	—
—	—	—	—

time step 2

e_{112}	e_{222}	e_{442}	e_{332}
e_{211}	e_{131}	e_{421}	e_{341}
—	—	—	—
—	—	—	—

time step 3

—	—	—	—
e_{212}	e_{132}	e_{422}	e_{342}
e_{231}	e_{411}	e_{141}	e_{321}
—	—	—	—

time step 4

—	—	—	—
—	—	—	—
e_{232}	e_{412}	e_{142}	e_{322}
e_{431}	e_{241}	e_{311}	e_{121}

time step 5

—	—	—	—
—	—	—	—
—	—	—	—
e_{432}	e_{242}	e_{312}	e_{122}

Table 7: Execution sequence of algorithm *MXk*

time step 1

e_{111}	e_{221}	e_{441}	e_{331}
—	—	—	—

time step 2

e_{112}	e_{222}	e_{442}	e_{332}
e_{211}	e_{131}	e_{421}	e_{341}

time step 3

e_{231}	e_{411}	e_{141}	e_{321}
e_{212}	e_{132}	e_{422}	e_{342}

time step 4

e_{232}	e_{412}	e_{142}	e_{322}
e_{431}	e_{241}	e_{311}	e_{121}

time step 5

—	—	—	—
e_{432}	e_{242}	e_{132}	e_{122}

Table 8: (*a*) TLT of algorithm *Ci*, (*b*) PAT of algorithm *Ci*

$k=$	1	2
$j=1$	1	2
2	2	3
3	3	4
4	4	1

$i=1$

$k=$	1	2
$j=1$	2	3
2	3	4
3	4	5
4	5	2

$i=2$

$k=$	1	2
$j=1$	3	4
2	4	5
3	5	6
4	6	3

$i=3$

$k=$	1	2
$j=1$	4	5
2	5	6
3	6	7
4	7	4

$i=4$

(*a*)

$\delta=$	1	2
$\gamma=1$	11	24
2	12	21
3	13	22
4	14	23

(*b*)

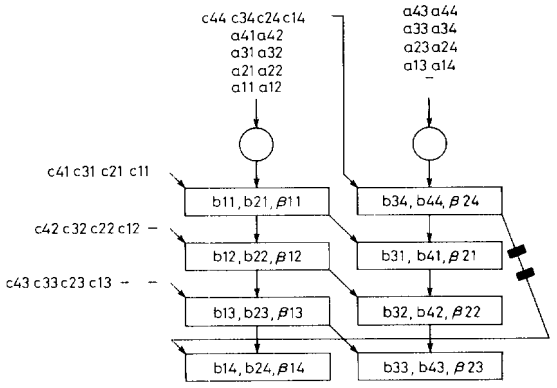


Fig. 8 *Another cylindrical array for the Winograd algorithm*

Table 9: (*a*) TLT of algorithm *Tk*, (*b*) PAT of algorithm *Tk*

$j=$	1	2	3	4
$i=1$	1	2	3	4
2	4	1	2	3
3	3	4	1	2
4	2	3	4	1

$k=1$

$j=$	1	2	3	4
$i=1$	4	1	2	3
2	3	4	1	2
3	2	3	4	1
4	1	2	3	4

$k=2$

(*a*)

$\delta=$	1	2	3	4
$\gamma=1$	11	12	13	14
2	21	22	23	24
3	31	32	33	34
4	41	42	43	44

(*b*)

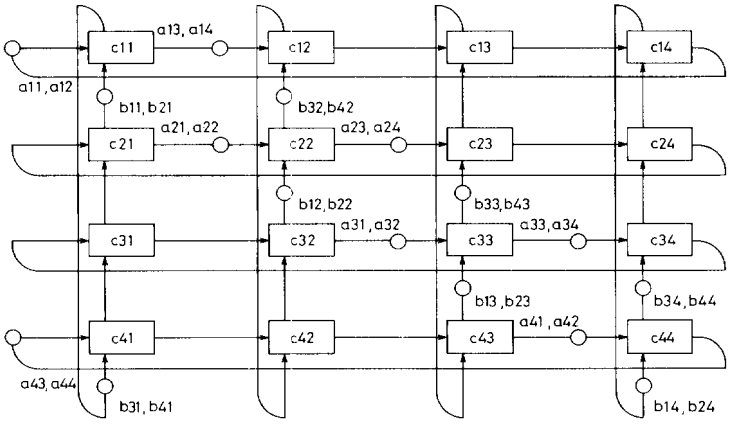
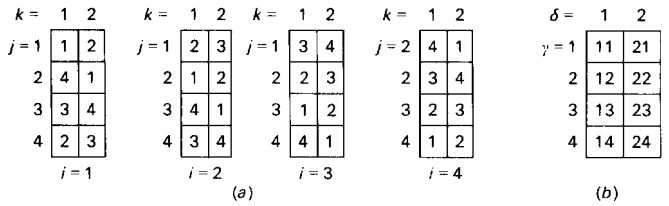


Fig. 9 *Torus array for the Winograd algorithm*

Table 10: (*a*) TLT of algorithm *Ti*, (*b*) PAT of algorithm *Ti*



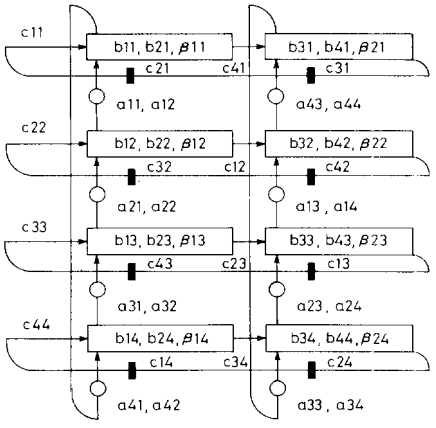


Fig. 10 *Another torus array for the Winograd algorithm*

3 Conclusion

We have proposed a unified approach for the design of parallel Winograd algorithms including a design proposed by Benaini and Robert [4], a similar design proposed by Jagadish and Kailath [5], and several novel designs. Results of comparisons of these algorithms are shown in Table 11. The results show that although systolic arrays (which execute systolic algorithms) have simpler wirings, their execution times are longer than the others and the utilisation of their processors are lower than the others. Nonplanar arrays, such as cylindrical array and torus array, have better performance as compared with systolic arrays. However, they have more complex wirings. Among these arrays, the torus array ***Ti*** is the most efficient one, because each processor of the array is fully utilised.

Table 11: Comparison of parallel Winograd algorithms

Algorithm	Execution time	Number of processors	Utilisation of processor
Sk	$5n/2-2$	$n \times n$	$n/2/(5n/2-2)$
Si	$5n/2-2$	$n \times n/2$	$n/(5n/2-2)$
Ck	$3n/2-1$	$n \times n$	$n/2/(3n/2-1)$
Xk	$3n/2-1$	$n \times n$	$n/2/(3n/2-1)$
MXk	$3n/2-1$	$n \times n/2$	$n/(3n/2-1$
Ci	$2n-1$	$n \times n/2$	$n/(2n-1)$
Tk	n	$n \times n$	1/2
Ti	n	$n \times n/2$	1

In Table 11, the estimation of time is based on the assumption that the operations are synchronised at the cell level. In near future, the proposed approach will be adopted to design parallel Winograd algorithms which are executed on arrays synchronised at operator level.

4 References

1 KUNG, H.T.: 'Why systolic architectures?', *Computer*, 1982, **15**, pp. 37–46
2 KUNG, S.Y.: 'VLSI array processor' (Prentice-Hall, Englewood Cliffs, NJ, 1988), Chapter 3
3 GUO-JIE, L., and WAH, B.: 'The design of optimal systolic arrays', *IEEE Trans. Comput.*, 1985, **C-34**, (1), pp. 66–77
4 BENAINI, A., and ROBERT, Y.: 'An even faster systolic array for matrix multiplication', *Parallel Computing*, 1989, **12**, pp. 249–254
5 JAGADISH, H.V., and KAILATH, T.: 'A family of new efficient arrays for matrix multiplication', *IEEE Trans. Comput.*, 1989, **38**, (1), pp. 149–155
6 HOROWITZ, E., and SAHNI, S.: 'Fundamentals of computer algorithms' (Computer Science Press, USA, 1987)
7 BARADA, H., and EL-AMAWAY, A.: 'A new methodology for mapping algorithms into VLSI arrays'. Proceedings of the 3rd annual parallel processing symposium, 1989, pp. 31–45
8 KAK, S.C.: 'Multilayered array computing'. Proceedings of 20th annual conf. on Information science and systems, Princeton, 1986, pp. 436–441
9 KAK, S.C.: 'A two-layered mesh array computing', *Parallel Computing*, 1988, **6**, pp. 383–385
10 KUNG, S.Y.: 'On supercomputing with systolic/wavefront array processors', *Proc. IEEE*, 1984, **72**, (7), pp. 867–884
11 PORTER, W.A., and ARAVENA, J.L.: 'Cylindrical arrays for matrix multiplication'. Proceedings of the 24th Annual Allerton Conference, October 1986, pp. 595–602
12 PORTER, W.A., and ARAVENA, J.L.: 'Orbital architectures with dynamic reconfiguration', *IEE Proc. E, Comput. Digit. Tech.*, 1987, **134**, (6), pp. 281–287
13 TSAY, J.C., and YUAN, S.: 'Some combinatorial aspects of parallel algorithm design for matrix multiplication', *IEEE Trans. Comput.*, 1992, **41**, (3), pp. 355–361
14 MEAD, C.A., and CONWAY, L.A.: 'Introduction to VLSI systems' (Addison Wesley, Reading, MA, 1980)
15 ARAVENA, J.L.: 'Triple matrix product architectures for fast signal processing', *IEEE Trans. Circuits Syst.*, 988, **CAS-35**, (1), pp. 119–123
16 ARAVENA, J.L., and BARBIR, A.O.: 'A class of low complexity high concurrence algorithms', *IEEE Trans. Parallel Distrib. Syst.*, 1991, **2**, (4), pp. 495–502
17 ZHANG, C.N., and YUN, D.: 'Multidimensional systolic networks for discrete Fourier transforms'. Proceedings of the international conference on Computer design, 1984, pp. 215–222
18 MA, Y.J., WANG, J.F., and LEE, J.Y.: 'Systolic array mapping of sequential algorithm for VLSI architecture'. Proceedings of international computer symposium, Tainan, Taiwan, ROC, 1986, pp. 865–874
19 DENES, J., and KEEDWELL, A.D.: 'Latin squares and their applications' (Academic Press, New York, 1974)
20 NAVARRO, J.J., LLABERIA, J.M., and VALERO, M.: 'Partitioning: an essential step in mapping algorithms into systolic array processors', *IEEE Computer*, July 1987, pp. 77–88