FILOMAT (Niš) 11 (1997), 109–117

# ABFT TECHNIQUE FOR MATRIX MULTIPLICATION ON SPACE–TIME OPTIMAL SYSTOLIC ARRAY

**Emina I. Milovanović, Ivan Z. Milentijević, Igor Ž. Milovanović and Teufik I. Tokić**

**Abstract.** A systematic design methodology which maps a matrix multiplication algorithm into fault tolerant systolic array is presented. The procedure used to obtain fault tolerant space–time optimal systolic array is an improved version of the data dependence method for designing of systolic arrays combined with Algorithm Based Fault Tolerance (ABFT) technique. The ABFT technique is used to detect and correct transient hardware faults. By the encoding scheme and preserving property of the matrix arithmetic, the resultant erroneous data produced by the faulty processing element can be detected and corrected.

## 1. Introduction

Fast matrix multiplication is highly demanded in signal and image processing, and scientific applications. The evolution in VLSI technology allows the implementation of matrix multiplication on a single VLSI chip using multiple regularly connected processing elements (PEs), which are known as systolic array (SA). The SAs achieve high performances because they provide high concurrency in computation.

Early SAs were designed in an ad–hoc manner. However, systematic design methodologies have recently been proposed to overcome the limitations of heuristic and designer intuition and to provide a formal framework for mapping algorithms into systolic architectures. Most of the proposed methodologies are based on concept of dependence vectors to order in time and space the index points representing the algorithm. The ordered index points are represented by nodes in a dependence graph with global dependencies and then, this graph is transformed into directed graph with local dependencies. The common characteristic of the methods based on the above

Received January 10, 1997; Revised October 10, 1997

1991 *Mathematics Subject Classification*: 60B20.

*Key words and phrases.* Systolic array, matrix multiplication, algorithm–based fault tolerance.

approach is that the same dependence graph is used for each allowable direction of the projection. As a consequence, the obtained SAs are not always optimal [1].

One of the most challenging problems in systolic processing is the designing of fault tolerant systolic architectures. Since each processor in such architecture contributes to the computation, any single temporary or permanent failure in a processor can break down an entire computing system. Therefore, reliability and fault tolerance have increasingly become inevitable important design issues [2]. Broadly speaking, fault tolerance schemes can be classified into two groups according to the type of failures they can deal with, that is production defects where faults occur due to flaws in circuit elements introduced during manufacture, and run–time (i.e. operational) defects where faults develop during the working life of a device. Although the operational defects have a considerably lower probability of occurrence than production defects, this faults cannot be ignored because the effects of their occurring can have much wider implications. An efficient approach for concurrent detection and correction of run–time errors is Algorithm Based Fault Tolerance (ABFT). It has been shown that ABFT is suitable for matrix arithmetic on SAs [3–6]. Huang and Abraham [3] have introduced the concept of ABFT for operational type failures in the form of permanent or transient hardware faults. The technique exploits information redundancy and has three main characteristics [7]:

1) data used by algorithm is encoded,
2) the algorithm is redesigned to operate on the encoded data and produces encoded output, and
3) redundancy in the encoding is used to detect and correct errors.

In this approach, some attribute of the function being performed by the VLSI SA is used, with some information of redundancy, to achieve high fault coverage with low overhead.

The objective of this paper is the synthesis of fault–tolerant processor–time optimal systolic array for matrix multiplication. We will present a systematic design methodology which maps a matrix multiplication algorithm to a fault tolerant systolic array. The design procedure, which is used to obtain the fault–tolerant processor time optimal SA for matrix multiplication, is an improved version of data dependence method for designing of SAs combined with ABFT technique. The result of our procedure is the SA for matrix multiplication with following features:

- the elements of input matrices $A$ and $B$, and resulting matrix $C$ are propagated through processing elements, i.e. no data item is resident in the array,

- the number of PEs, for input matrices of order $n \times n$, is $n^2 + n$,
- total execution time, for the same case, is $4n - 1$,
- the array tolerates single transient/permanent hardware faults.

## 2. A brief survey of the ABFT

In the text that follows we will briefly describe the ABFT technique for matrix multiplication proposed by Huang and Abraham [3].

Let $A = [a_{ij}]$ be an $n \times n$ matrix and $\vec{e} = [1 \quad 1 \quad \cdots \quad 1]^T$ an $n \times 1$ encoding vector. Based on the encoding vector the matrix $A$ can be encoded as either column encoded matrix, $A_c$, row encoded matrix, $A_r$, or full encoded matrix, $A_f$, in the following way

$$A_c = \left[\begin{array}{c} A \\ \hline \vec{e}^T A \end{array}\right], \tag{1}$$

where

(2)

$$a_{n+1,j} = (A_c)_{n+1,j} = [1 \quad 1 \quad \cdots \quad 1] \begin{bmatrix} a_{1j} \\ \vdots \\ a_{nj} \end{bmatrix} = \sum_{i=1}^{n} a_{ij}, \quad j = 1, \ldots, n,$$

is a column checksum vector;

$$A_r = [A \mid A\vec{e}], \tag{3}$$

where

$$a_{i,n+1} = (A_r)_{i,n+1} = [a_{i1} \quad \cdots \quad a_{in}] \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} = \sum_{j=1}^{n} a_{ij}, \quad i = 1, \ldots, n \tag{4}$$

is a row checksum vector;

$$A_f = \left[\begin{array}{c|c} A & A\vec{e} \\ \hline \vec{e}^T A & \vec{e}^T A\vec{e} \end{array}\right]. \tag{5}$$

For the product of two $n \times n$ matrices $A$ and $B$ the following implication is valid

$$A \times B = C \implies A_c \times B_r = C_f = \left[\begin{array}{c|c} AB & AB\vec{e} \\ \hline \vec{e}^T AB & \vec{e}^T AB\vec{e} \end{array}\right]. \tag{6}$$

Fault–tolerant matrix multiplication on the systolic array can be implemented as a three stage process [7]:

1) Error detection;
2) Error location;
3) Error correction.

**Error detection:** Assume that $\vec{e}^T A$ and $B\vec{e}$ are known and compute $\vec{c} = \vec{e}^T AB$, $\vec{r} = AB\vec{e}$ and $A_c \times B_r$. Denote by $\vec{c}'$ and $\vec{r}'$ the first $n$ elements of the last row and last column of $C_f$, respectively. An error is indicated in $C$ if $\vec{g}_c = \vec{c} - \vec{c}' \neq 0$ and/or $\vec{g}_r = \vec{r} - \vec{r}' \neq 0$.

**Error location:** A map of the error locations is given by the outer product matrix $(\vec{g}_c \vec{g}_r)^T$ where

$$(g_c g_r)_{ij} = \begin{cases} 1 & \text{iff } g_c(i) g_r(j) \neq 0\,, \\ 0 & \text{otherwise}\,. \end{cases} \tag{7}$$

**Error correction:** The erroneous values are corrected by:

a) $c_{ij} = \begin{cases} c_{ij} \pm g_r(j)\,, \text{ or} \\ c_{ij} \pm g_c(i)\,, \end{cases}$ iff $(\vec{g}_c \vec{g}_r)^T_{ij} = 1\,.$

b) By replacing $\vec{r}$ with $\vec{r}'$ and $\vec{c}$ with $\vec{c}'$ in the case of checksum vector error.

Let us note that the described scheme can correct only a single fault in the $C_f$.

## 3. The synthesis of space–time optimal systolic array for matrix multiplication

In this section we will give formulas for synthesizing space–time optimal systolic array for multiplication of rectangular matrices.

Let $A = [a_{ik}]$ and $B = [b_{kj}]$ be an $m \times n$ and $n \times p$ matrices, respectively. Their product $C = A \times B$ can be computed by the following equation

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}\,, \quad i = 1, \ldots, m, \quad j = 1, \ldots, p\,.$$

For the elements $a$, $b$ and $c$ we involve the following periodicity

$$\begin{aligned} a(i,k) &\equiv a(i,0,k) \equiv a(i+m,0,k) \equiv a(i,0,k+n) \equiv a_{ik}\,, \\ b(k,j) &\equiv b(0,j,k) \equiv b(0,j+p,k) \equiv b(0,j,k+n) \equiv b_{kj}\,, \\ c(i,j) &\equiv c(i,j,0) \equiv c(i+m,j,0) \equiv c(i,j+p,0) \equiv c_{ij}\,. \end{aligned} \tag{8}$$

Now, denote by $p(i,j,k) \rightarrow \begin{bmatrix} x \\ y \end{bmatrix}$ the $(x,y)$ position of the processing element in the projection plane. Similarly, $a(i,0,k) \rightarrow \begin{bmatrix} x \\ y \end{bmatrix}_a$, $b(0,k,j) \rightarrow \begin{bmatrix} x \\ y \end{bmatrix}_b$, and $c(i,j,0) \rightarrow \begin{bmatrix} x \\ y \end{bmatrix}_c$ denote the $(x,y)$ positions for elements $a$, $b$ and $c$, respectively.

The $(x,y)$ positions of processing elements in the projection plane are obtained according to the following:

**for** $k := 1$ **to** $n$
  **for** $j := 1$ **to** $p$
    **for** $i := 1$ **to** $m$

$$p(i,j,i+k-1) \rightarrow \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1-k \\ j \end{bmatrix} \tag{9}$$

    **endfor;**
  **endfor;**
**endfor.**

The $(x,y)$ positions of matrix elements in the projection plane, are obtained by the following formulas:

**for** $k := 1$ **to** $n$
  **for** $j := 1$ **to** $p$
    **for** $i := 1$ **to** $m$

$$\begin{aligned} c(2i-1,j,0) &\rightarrow \begin{bmatrix} x \\ y \end{bmatrix}_c = \begin{bmatrix} 2i+j-2 \\ j \end{bmatrix} + rm \begin{bmatrix} -1 \\ 0 \end{bmatrix} \\ a(2i-1,0,i+k-1) &\rightarrow \begin{bmatrix} x \\ y \end{bmatrix}_a = \begin{bmatrix} 1-k \\ 3-2i-k \end{bmatrix} + rm \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ b(0,j,i+k-1) &\rightarrow \begin{bmatrix} x \\ y \end{bmatrix}_b = \begin{bmatrix} 4-2i-j-2k \\ j \end{bmatrix} + rm \begin{bmatrix} 1 \\ 0 \end{bmatrix} \end{aligned} \tag{10}$$

    **endfor;**
  **endfor;**
**endfor;**

when $m$ is odd, and

**for** $k := 1$ **to** $n$
  **for** $j := 1$ **to** $p$
    **for** $i := 1$ **to** $\left[\frac{m}{2}\right]$

$$c(2i-1,j,0) \rightarrow \begin{bmatrix} x \\ y \end{bmatrix}_c = \begin{bmatrix} 2i+j-2 \\ j \end{bmatrix} + r(m-1) \begin{bmatrix} -1 \\ 0 \end{bmatrix}$$

$$a(2i-1,0,i+k-1) \rightarrow \begin{bmatrix} x \\ y \end{bmatrix}_a = \begin{bmatrix} 1-k \\ 3-2i-k \end{bmatrix} + r(m-1)\begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$b(0,j,i+k-1) \rightarrow \begin{bmatrix} x \\ y \end{bmatrix}_b = \begin{bmatrix} 4-2i-j-2k \\ j \end{bmatrix} + r(m-1)\begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

**endfor;**

(11) **for** $i := \left[\frac{m}{2}\right] + 1$ **to** $m$

$$c(2i,j,0) \rightarrow \begin{bmatrix} x \\ y \end{bmatrix}_c = \begin{bmatrix} 2i+j-2 \\ j \end{bmatrix} + r(m-1)\begin{bmatrix} -1 \\ 0 \end{bmatrix}$$

$$a(2i,0,i+k-1) \rightarrow \begin{bmatrix} x \\ y \end{bmatrix}_a = \begin{bmatrix} 1-k \\ 3-2i-k \end{bmatrix} + r(m-1)\begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$b(0,j,i+k-1) \rightarrow \begin{bmatrix} x \\ y \end{bmatrix}_b = \begin{bmatrix} 4-2i-j-2k \\ j \end{bmatrix} + r(m-1)\begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

**endfor;**

**endfor;**

**endfor;**

when $m$ is even. In the following formulas $r$ is the maximal integer from the set $\{0,1\}$ which satisfies the condition

$$-2(i-1) + r\bar{m} < 0\,, \quad \text{if } i = 1 \text{ then } r = 0\,, \tag{12}$$

where

$$\bar{m} = \begin{cases} m\,, & \text{for } m \text{ odd}\,, \\ m-1\,, & \text{for } m \text{ even}\,. \end{cases}$$

The systolic array for matrix multiplication synthesized according to the given formulas has $\Omega = n \times p$ processing elements, active execution time $T_{exe} = m + n + p - 2$ time units, and total execution time, $T_{tot}$, which includes times for data input, output and execution, of $2n + m + p - 3$ time units. Here, the duration of add–multiply operation is taken as a time unit. Compared with the results given in the literature, the array size is reduced significantly, e.g. $\Omega = p(m+n-1)$ found in [8], without increasing execution time or PE's complexity.

Since each component $c_{ij}$ of the resulting matrix $C$, consists of $n$ partial products, it is obvious that optimal SA should have $n$ PEs along one dimension. The another requirement is that the resulting matrix is obtained in row–major ordering, so the second dimension of the SA has to be equal to the number of columns, i.e. $p$. Under this conditions, the array obtained by (9), (10) and (11), is optimal.

Data flow in the array obtained according to (9) and (10) (i.e. (11)) for the case $m = 4$, $n = 3$ is presented in Fig. 1.

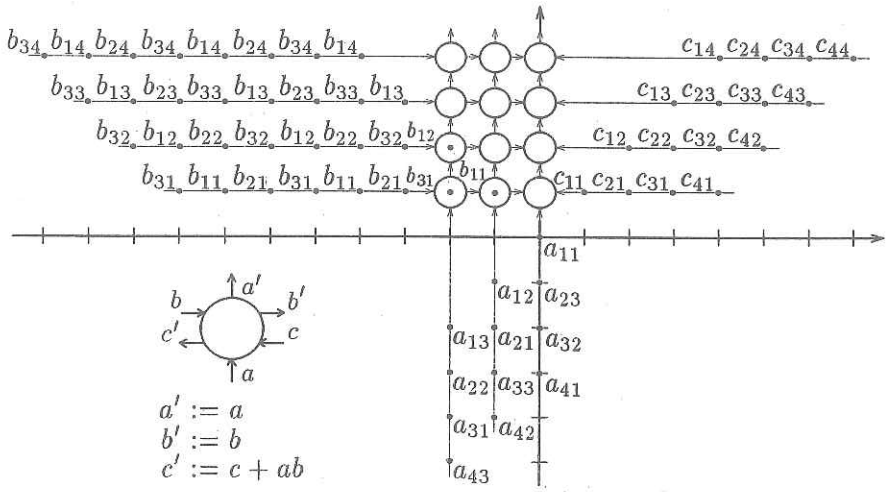


Figure 1. Data flow in space–time optimal SA for matrix multiplication for $m = 4$, $n = 3$, $p = 4$

When $m = n = p$ the number of PEs, active execution time, and efficiency of the SA obtained by the proposed procedure are

$$\Omega = n^2, \quad T_{exe} = 3n - 2, \quad E = \frac{n}{3n - 2},$$

respectively. For the same case, the corresponding values for the best array find in [8] are

$$\Omega = 2n^2 - n, \quad T_{exe} = 3n - 2, \quad E = \frac{n^2}{(2n - 1)(3n - 2)}.$$

| $n$ | standard | new |
|---|---|---|
| 5 | 45 | 25 |
| 10 | 190 | 100 |
| 100 | 19900 | 10000 |
| $\infty$ | 100% | 50% |

Table 1. Number of processors for two systolic arrays for the matrix multiplication algorithm

| $n$ | standard | new |
|---|---|---|
| 5 | 21.4% | 38.5% |
| 10 | 18.8% | 35.7% |
| 100 | 16.9% | 33.6% |
| $\infty$ | 16.7% | 33.3% |

Table 2. Efficiency fotr the compared systolic arrays for the matrix multiplication algorithm

Tables 1. and 2. summarise the characteristics of the new array obtained by the procedure proposed in this paper, and the arry described in [8] for $n = 5$, $n = 10$, $n = 100$, and $n \to \infty$.

## 4. ABFT technique and optimal array

In order to obtain the space–time optimal FT systolic array for matrix multiplication, we will combine the ABFT technique with the formulas for synthesizing space–time optimal SA for multiplication of rectangular matrices. Namely, if in formulas (9), (10) (i.e. (11)) and (12) we replace $m$ and $p$ with $n+1$ and assume that elements $a_{n+1,k}$ and $b_{k,n+1}$, $k = 1, \ldots, n$, are defined by (2) and (4), respectively, then the resulting matrix will represent the full encoded matrix $C_f$. Having this in mind, Fig. 1 can be considered as FT array for multiplication of two $3 \times 3$ matrices. This is clarified in Fig. 2.

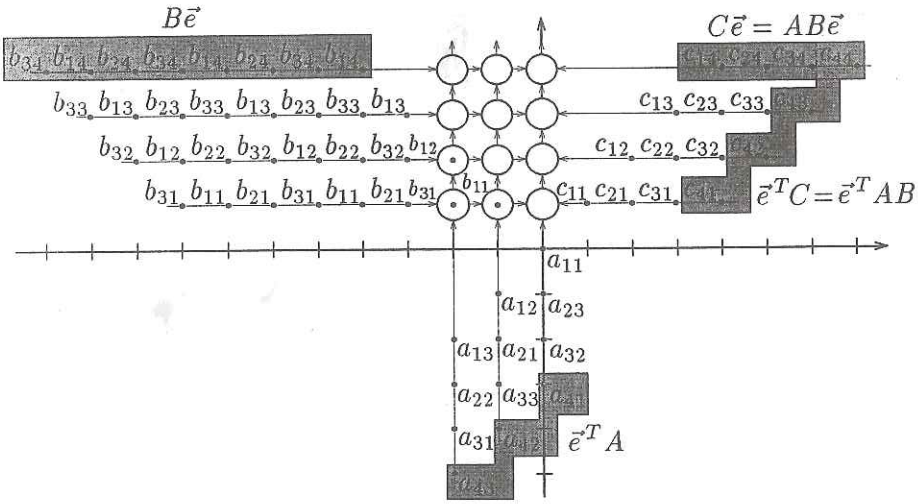


Figure 2. Data flow in the fault–tolerant array for $n = 3$

Shaded areas represent row and column encoded vectors. The activities that are performed during FT matrix multiplication are presented in Table 3.

| | Activity | Computation | Who works |
|---|---|---|---|
| 1 | pre–processing | $\vec{e}^T A$, $B\vec{e}$ | host or array |
| 2 | processing | $C_f = A_c B_r$ | array |
| 3 | error detection | $\vec{c}' = \vec{e}^T C$, $\vec{g}_c = \vec{c} - \vec{c}'$<br>$\vec{r}' = C\vec{e}$, $\vec{g}_r = \vec{r} - \vec{r}'$ | host or array |
| 4 | error correction | if $(g_c g_r)_{ij}^T = 1$ then<br>$c_{ij} = c_{ij} \pm g_r(j)$ or<br>$c_{ij} = c_{ij} \pm g_c(i)$ | host |

Table 3. The activities during FT matrix multiplication

From Table 3 one can see that matrix multiplication is always performed in the systolic array, and error correction by the host. The activities 1 and 3 can be performed either by the array or by the host computer.

## References

[1] I. Ž. Milovanović, E. I. Milovanović, I. Z. Milentijević, M. K. Stojčev, *Designing of Processor–Time Optimal Systolic Arrays for Band Matrix–Vector Multiplication,* Computers Math. Appl. **32** (2) (1996), 21–31.

[2] M. O. Esonu, A. J. Al-Khalili, S. Hariri, D. Al-Khalili, *Fault–Tolerant Design Methodology for Systolic Array Architectures,* IEE Proc. Comput. Digit. Tech. **141** (1) (1994), 17–28.

[3] K. H. Huang, J. A. Abraham, *Algorithm Based Fault Tolerance for Matrix Operations,* IEEE Trans. Comp. **C-33** 6 (1984), 518–528.

[4] P. Banerjee, J. A. Abraham, *Bounds on Algorithm Based Fault Tolerance in Multiple Processor Systems,* IEEE Trans. Comput. **C-35** 4 (1986), 296–306.

[5] J. Y. Jou, J. A. Abraham, *Fault–Tolerant Matrix Arithmetic and Signal Processing on Highly Concurrent Computing Structures,* Proc. IEEE **5** (1986), 732–740.

[6] F. T. Luk, *Algorithm Based Fault Tolerance for Parallel Matrix Equation Solvers,* SPIE (564) Real Time Signal Processing, VIII, 1985, 49–53.

[7] G. M. Megson, D. J. Evans, *Algorithmic Fault Tolerance for Matrix Operations on Triangular Arrays,* Parallel Comput. **10** (1989), 207–219.

[8] D. I. Moldovan, *Parallel Processing: From Applications to Systems,* Morgan Kaufmann Publ, San Mateo, CA, 1993.

University of Niš, Faculty of Electronic Engineering, P.O. Box 73, Beogradska 14, 18000 Niš, Serbia