CHECKSUM SCHEMES FOR FAULT TOLERANT SYSTOLIC COMPUTING

R.P. Brent
(Australian National University, Australia)

and

F.T. Luk and C.J. Anfinson
(Cornell University, New York)

CHECKSUM SCHEMES FOR FAULT TOLERANT SYSTOLIC COMPUTING

R.P. Brent
(Australian National University, Australia)

and

F.T. Luk and C.J. Anfinson
(Cornell University, New York)

ABSTRACT

The weighted checksum scheme has been proposed as a low-cost fault tolerant procedure for parallel matrix computations. To guarantee multiple error detection and correction, the chosen weight vectors must satisfy some very specific properties about linear independence. We will provide a theoretical framework for these properties, and prove that for a distance d+1 scheme, if a maximum of $\lfloor d/2 \rfloor$ errors ensue, the exact number of errors can be determined. We will derive a procedure for correcting the errors. Previous weight generating methods that fulfil the independence criteria have troubles with numerical overflow. We will present a new scheme that generates weight vectors to meet the requirements about independence and to avoid the difficulties with overflow.

1. INTRODUCTION

The importance of solving signal processing problems in real time and the development of VLSI and wafer-scale technology have led to research in systolic arrays and algorithms. There is a need for high-performance digital signal processing systems which are extremely reliable. Algorithm-based fault tolerance has been proposed to meet this reliability need since the most common alternative of duplicating hardware is often too expensive to be practical.

The weighted checksum scheme, originally developed by Abraham and students [6], [7], provides low-cost error protection for applications that include matrix addition, matrix multiplication, and triangular decompositions (see also [8] and [10]). We will present a theoretical framework for the scheme and show how to decide upon the exact number of errors. Techniques for error correction were known only for the cases of one error [7] and two errors [1]. We will present a scheme for correcting [d/2] errors. The previously proposed weights are powers of integers, and thus can become very large. We will propose a technique for generating reasonably sized weights.

This paper is organized as follows. We first review the weighted checksum scheme and discuss an important theorem that guarantees multiple error detection and correction. Then we prove that, if a maximum of [d/2] errors ensue in a distance d+1 code, the errors can be detected and the exact number of errors determined. Furthermore, we show how we can always correct the errors and present a procedure for doing so. Lastly, we discuss the problem of numerical overflow and propose a new method for weight generation that overcomes this difficulty.

2. BACKGROUND

In [1], [6] and [7], a linear algebraic model of the weighted checksum scheme is developed, allowing parallels to be drawn between algorithm-based fault tolerance and coding theory. We briefly review this important background material, and discuss the fault model relevant to our results, as well as several important assumptions and their implications.

2.1 Definitions

We define a $d \times (n+d)$ consistency check matrix H by

$$H = \begin{bmatrix} \Psi_0^0 & \Psi_1^0 & \cdots & \Psi_{n-1}^0 & 1 & 0 & \cdots & 0 \\ \Psi_0^1 & \Psi_1^1 & \cdots & \Psi_{n-1}^1 & 0 & 1 & \cdots & 0 \\ \cdot & \cdot & \cdots & \cdot & 0 & 0 & \cdots & 0 \\ \cdot & \cdot & \cdots & \cdot & \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot & \cdot & \cdot & \cdots & \cdot \\ \Psi_0^{d-1} & \Psi_1^{d-1} & \cdots & \Psi_{n-1}^{d-1} & 0 & 0 & \cdots & 1 \end{bmatrix} \qquad (2.1)$$

It is proved in [1] that for $\Psi_i \neq \Psi_j$, where $i \neq j$, every set of d columns of H is linearly indpendent. This matrix is

similar to the parity check matrix in coding theory and is said to generate the code. In order to make precise the meaning of distance in the code space, we define a metric upon the domain of H. It is easily checked that the metric satisfies the properties of a distance [4].

Definition 2.1. The code space C of H consists of all vectors which lie in the null space N(H) of H, where $N(H) = \{x : Hx = 0\}$.

Definition 2.2. The distance between two vectors v and w in the domain of H, *dist* (v,w), equals the number of components in which v and w differ.

Definition 2.3. The distance of the code space C is the minimum of the distances between all possible pairs of distinct vectors in N(H); i.e., distance of C = min {*dist* (v,w) : v,w in N(H), $v \neq w$}.

Definition 2.4. Let x be in N(H), and $\hat{x}$ be a possibly erroneous version of x. Define the syndrome vector s by $s = H\hat{x}$, and the correction vector c by $c = \hat{x} - x$.

Note that $Hc = s$ since $Hc = H\hat{x} - Hx = H\hat{x} = s$. For simplicity we will subscript the vectors c and s as follows:

$$c = (c_0, c_1, \ldots, c_{n+d-1})^T$$

and

$$s = (s_0, s_1, \ldots, s_{d-1})^T.$$

We can now rigorize the definitions of error detection and correction. Let α denote the total number of errors that have occurred and define γ by

$$\gamma = \left[\frac{d}{2} \right]. \qquad (2.2)$$

A coding system can *detect* α errors if the syndrome vector s is nonzero whenever $1 \leq \alpha \leq d$. A coding scheme can *correct* α errors if $\hat{x}$ can be corrected to x for $1 \leq \alpha \leq \gamma$. The next result from [1] states the sufficient conditions for error detection and correction.

Theorem 2.1. If every set of d columns of H is linearly independent, then the distance of the code is d+1, a maximum of d errors can be detected, and a maximum of γ errors can be corrected. □

2.2 *Fault model and assumptions*

As we are primarily interested in multiprocessor systems for real time digital signal processing (e.g., systolic arrays), we assume that a module (a processor or computational unit) makes arbitrary logical errors in the event of a fault and that the system is periodically checked for permanent and intermittent errors. We focus our attention on soft errors, and suppose that the soft error rate is small under normal operating conditions, which is reasonable since it has been reported that the soft error rate for a large VLSI chip ($1\ \mathrm{cm}^2$) is 10^{-4} per hour [12].

Two major assumptions are made in this paper. First, for our correction procedure, we assume that no errors occur in the checksums themselves. It should be noted that for error detection this assumption is not needed. We also assume that $1 \le \alpha \le \gamma$.

3. ERROR DETECTION

In this section we discuss a matrix factorization that allows us to determine the exact value of α. This a priori knowledge of α will prove important later in the correction procedure.

3.1 *The syndrome vector*

Consider the syndrome vector s. By assuming that no errors occur in the checksums themselves, we get $c_n = c_{n+1} = \dots = c_{n+d-1} = 0$. Hence, the jth element of s is given by the formula

$$s_j = \sum_{k=0}^{n-1} \Psi_k^j c_k, \text{ for } j = 0, 1, \dots, d-1.$$

Clearly there are only α nonzero elements in the set $\{c_0, c_1, \dots, c_{n-1}\}$, because by definition $c = \hat{x} - x$, and $\hat{x}$ differs from x by α elements. Denote the nonzero elements of c by c_{i_k}, for $k = 1, \dots, \alpha$. Hence, s_j may be expressed as

$$s_j = \sum_{k=1}^{\alpha} \Psi_{i_k}^j c_{i_k}, \text{ for } j = 0, 1, \dots, d-1. \tag{3.1}$$

We would like to express s_j in terms of known parameters, and at this time α is unknown and γ is known. So, let

$$y_k = c_{i_k} \text{ and } \xi_k = \Psi_{i_k}, \text{ for } k = 1, \dots, \alpha,$$

and

$$y_k = 0 \text{ and } \xi_k = \Psi_j, \text{ for } k = \alpha+1, \dots, \gamma,$$

such that $\xi_i \ne \xi_\ell$ for all $i \ne l$, where $i, \ell = 1, \dots, \gamma$. With these substitutions s_j becomes

$$s_j = \sum_{k=1}^{\gamma} \xi_k^j y_k, \text{ for } j = 0, 1, \dots, d-1. \tag{3.2}$$

3.2 *Find number of errors*

Define a $\gamma \times \gamma$ symmetric Hankel matrix by

$$K = \begin{bmatrix} s_0 & s_1 & \cdots & s_{\gamma-1} \\ s_1 & s_2 & \cdots & s_\gamma \\ . & . & \cdots & . \\ . & . & \cdots & . \\ . & . & \cdots & . \\ s_{\gamma-1} & s_\gamma & \cdots & s_{2\gamma-2} \end{bmatrix} \tag{3.3}$$

Theorem 3.1. The matrix K has the factorization $K = VYV^T$, where $Y = \mathit{diag}[y_1, y_2, \dots, y_\gamma]$ and

$$V = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \xi_1 & \xi_2 & \cdots & \xi_\gamma \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \xi_1^{\gamma-1} & \xi_2^{\gamma-1} & \cdots & \xi_\gamma^{\gamma-1} \end{bmatrix}.$$

Furthermore, the matrix has rank α, where α denotes the number of errors that have occurred.

Proof. Multiplying out VYV^T we find that the (i,j) entry is equal to s_{i+j-2}, and thus $(K)_{ij}$. The second part of our claim follows from the observation that $rank(K) = rank(Y)$. □

An accurate, albeit expensive, way to determine the rank of the symmetric matrix K is to compute its eigenvalue decomposition. As will be seen, we will need a nonsingular matrix for the correction procedure to work effectively. The following corollary to Theorem 3.1 will allow us to handle the case of a rank deficient K. Define K_α as the leading $\alpha \times \alpha$ principal submatrix of K:

$$K_\alpha = \begin{bmatrix} s_0 & s_1 & \cdots & s_{\alpha-1} \\ s_1 & s_2 & \cdots & s_\alpha \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ s_{\alpha-1} & s_\alpha & \cdots & s_{2\alpha-2} \end{bmatrix}. \tag{3.4}$$

Corollary 3.1. The $\alpha \times \alpha$ matrix K_α has full rank, and can be factored as $K_\alpha = V_\alpha Y_\alpha V_\alpha^T$, where $Y_\alpha = diag[y_1, y_2, \ldots, y_\alpha]$ and

$$V_\alpha = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \xi_1 & \xi_2 & \cdots & \xi_\alpha \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \xi_1^{\alpha-1} & \xi_2^{\alpha-1} & \cdots & \xi_\alpha^{\alpha-1} \end{bmatrix}.$$

4. ERROR CORRECTION

In this section we outline our correction procedure. We will assume that α has been determined such that K_α is nonsingular.

4.1 A polynomial

Define a polynomial $P(z)$ whose roots are the unknown weights ξ_i, for $i = 1, \ldots, \alpha$. We will show that we can determine the coefficients of the polynomial by solving a linear system involving the matrix K_α.

$$P(z) = \prod_{i=1}^{\alpha} (z - \xi_i) = \sum_{i=0}^{\alpha} a_i z^i, \tag{4.1}$$

where the coefficients of P are given by

$$a_i = (-1)^{i+\alpha} \sum_{j_1<\cdots<j_{\alpha-i}} \xi_{j_1} \cdots \xi_{j_{\alpha-i}}, \tag{4.2}$$

and $a_\alpha = 1$. The next theorem and its two corollaries will lay a foundation for the correction procedure. We find necessary the following definitions. First, define A_{-k} as the matrix A with the kth column deleted and

$$a = (a_0, a_1, \ldots, a_\alpha)^T.$$

Then, let

$$f = (s_\alpha, s_{\alpha+1}, \ldots, s_{2\alpha-1})^T$$

and

$$F = (K_\alpha, f) = \begin{bmatrix} s_0 & s_1 & \cdots & s_{\alpha-1} & s_\alpha \\ s_1 & s_2 & \cdots & s_\alpha & s_{\alpha+1} \\ \cdot & \cdot & \cdots & \cdot & \cdot \\ \cdot & \cdot & \cdots & \cdot & \cdot \\ \cdot & \cdot & \cdots & \cdot & \cdot \\ s_{\alpha-1} & s_\alpha & \cdots & s_{2\alpha-2} & s_{2\alpha-1} \end{bmatrix}.$$

Writing out the jth element of the matrix-vector product Fa, we get

$$(Fa)_j = \sum_{k=1}^{\alpha} \xi_k^j y_k P(\xi_k) = 0, \text{ for } j = 0,1,\ldots,d-1.$$

Theorem 4.1. The coefficients of the polynomial $P(z)$ satisfy the equation

$$Fa = 0. \quad \square \tag{4.3}$$

Corollary 4.1. Let $D_\alpha = \det(K_\alpha)$. For $k = 0,1,\ldots,\alpha-1$, the coefficient a_k can be computed by the formula

$$a_k = (-1)^{k+\alpha} D_k / D_\alpha, \tag{4.4}$$

where $D_k = \det(F_{-k})$. $\square$

Corollary 4.2. Define the polynomial $Q(z)$ as $\sum_{i=0}^{\alpha} (-1)^i D_i z^i$. Then $Q(z)$ has roots $\xi_1, \xi_2, \ldots, \xi_\alpha$. $\square$

4.2 *Correction procedure*

Thus, in order to find the unknown weights, we need to find the vector a of coefficients of the polynomial $P(z)$. So, we may first solve (4.3) for a, and then find the roots of $P(z)$. Alternatively, we could compute the determinants D_ks and find the roots of $Q(z)$, which will give us the unknown weights by Corollary 4.2. Using the above facts, there are four steps that need to be accomplished for correction.

Procedure

(1) Find *rank* (K). This can be done by finding the eigenvalues of the symmetric matrix K.

(2) Solve for the coefficients $(a_0, a_1, \ldots, a_\alpha)$ of $P(z)$. Note that this can be accomplished by either solving the system (4.3) or by using equation (4.4). This step requires K_α to be of full rank.

(3) Find the roots $(\xi_1, \ldots, \xi_\alpha)$ of $P(z)$ or $Q(z)$. Note that these can be computed as the eigenvalues of a certain companion matrix. This will give us the vector ξ.

(4) Find the vector $y = (y_1 y_2, \ldots, y_\alpha)^T$ by solving the systems $\Xi y = s$, where

$$\Xi = \begin{bmatrix} \xi_1^0 & \xi_2^0 & \cdots & \xi_\alpha^0 \\ \xi_1^1 & \xi_2^1 & \cdots & \xi_\alpha^1 \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \xi_1^{d-1} & \xi_2^{d-1} & \cdots & \xi_\alpha^{d-1} \end{bmatrix}.$$

Actually Ξ is a nonsingular Vandermonde matrix and there are fast techniques available to solve systems of linear equations involving a Vandermonde matrix [4]. Note that solving the system $\Xi y = s$ is equivalent to solving $Hc = s$ for the nonzero values of c, and that our procedure is very similar to the Reed-Solomon code [11]. Our procedure details the mathematical analysis of the problem, and can use further improvement as a numerical algorithm. The procedure will not be prohibitively expensive to implement. Furthermore, one may use a systolic array, or some other parallel machine which can solve the eigenvalue problem rapidly [2], [13]; since $d<n$, the size of the eigenvalue problem we need to solve is much smaller than that of the original problem.

5. AVOIDING OVERFLOWS

We present a new technique for generating weight vectors which satisfy the conditions of Theorem 2.1, and avoid the weight overflow problem which plagues previous schemes. Big weights introduce large roundoff errors that cannot be distinguished from true errors caused by faults (see [9] for a discussion.) Our scheme is dependent on the values of n and d, but as the step of generating the weight vectors is pre-processing, it will not slow down the execution of the algorithm. Our method uses modular arithmetic, but not in the same way as suggested by Abraham et al. [6], [7].

5.1 Modular arithmetic

Consider the values of n and d to be known and fixed. The size of the input matrix determines n, and d is selected by the user according to a model which decides on the expected number of errors. Choose a prime number p such that $p > n+d$. It is well known that for any prime p there exists a finite field of p elements [5]. Let α be a primitive element of the finite field of p elements. Recall that a primitive element β of a finite field with q elements is an element with order $q-1$, i.e., the smallest integer m for which $\beta^m = 1 \; mod \; q$ is $m = q-1$. It is easy to see that powers of β generate all the nonzero elements of the field.

Let B be the following matrix

$$B = (b_{ij}) = (\alpha^{(i-1)(j-1)}), \text{ for } 1 \le i \le d \text{ and } 1 \le j \le n+d. \qquad (5.1)$$

Partition B as

$$B = [V|M] \qquad (5.2)$$

Here V has dimension dxn and W has dimension dxd. Now, in order to get the desired form of H, we multiply the matrix B *mod* p by the matrix W^{-1} *mod* p. That is

$$H = W^{-1}[V|M] \; mod \; p = [W^{-1}V \; mod \; p|I]. \qquad (5.3)$$

To make the above procedure valid, we must prove the following. First, we need to show that W is nonsingular so that the multiplication in (5.3) is valid. Then we need to prove that every set of d columns of the resulting matrix H is linearly independent, so that we can guarantee the detection of a maximum of d errors. A proof of the next theorem can be found in [3].

Theorem 5.1. The matrix B *mod* p, where B is given by equation (5.2), satisfies the condition that every set of d columns is linearly independent. □

Corollary 5.1. The dxd matrix W *mod* p, where W is defined in equation (5.2), is nonsingular. □

Corollary 5.2. The matrix H, given by equation (5.3), satisfies the hypotheses of Theorem 2.1. □

We note that the choice of W is by no means unique. We could choose any d columns of B and permute the matrix into the form given by equation (5.2).

5.2 An example

Suppose we are given a problem where $n = 8$ and d is selected to be 4. Thus, by fulfilling the requirements of Theorem 2.1 we will be able to detect a maximum of 4 errors in a 12-element encoded vector. We take $p = 13$, and $\alpha = 2$ is a primitive root of p. The matrix B *mod* p has the form

$$B \; mod \; p = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 & 3 & 6 & 12 & 11 & 9 & 5 & 10 & 7 \\ 1 & 4 & 3 & 12 & 9 & 10 & 1 & 4 & 3 & 12 & 9 & 10 \\ 1 & 8 & 12 & 5 & 1 & 8 & 12 & 5 & 1 & 8 & 12 & 5 \end{bmatrix}.$$

Now we need to find the inverse of W *mod* p. If we use the Gauss-Jordan method for finding matrix inverses, we can do the inversion *mod* p since the procedure consists of only elementary row operations. We then perform the multiplication $W^{-1}V$ *mod* p. Alternatively, we can find the LU factors of W via Gaussian elimination *mod* p, which also involves only elementary row operations, and then compute each column of the product $W^{-1}V$ *mod* p using the LU factors. While computing $W^{-1}V$ *mod* p is expensive, we once again point out that it is all pre-processing; it will not add extra running time to the algorithm. Hence, we can find the inverse of W *mod* p,

$$W^{-1}\ mod\ p = \begin{bmatrix} 8 & 5 & 6 & 8 \\ 6 & 11 & 0 & 12 \\ 5 & 0 & 9 & 7 \\ 8 & 10 & 11 & 12 \end{bmatrix}.$$

Multiplying out $W^{-1}B\ mod\ p$, we get the desired matrix

$$H = \begin{bmatrix} 1 & 2 & 12 & 4 & 7 & 6 & 1 & 10 & 1 & 0 & 0 & 0 \\ 3 & 7 & 12 & 11 & 12 & 12 & 9 & 5 & 0 & 1 & 0 & 0 \\ 8 & 6 & 12 & 5 & 2 & 8 & 7 & 11 & 0 & 0 & 1 & 0 \\ 2 & 12 & 4 & 7 & 6 & 1 & 10 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

5.3 *Comparison with previous work*

We examine how our method differs from these previous schemes. For means of comparison we consider fixed point arithmetic since it is used in [6] and [7]. Let r denote the word length. Huang and Abraham [6] suggested that we compute the actual checksum values modulo 2^r. In Jou and Abraham [7], a similar technique is proposed where the checksum column is computed modulo 2^r and the weighted checksum column is computed modulo N, where N is the largest prime less than 2^{r+1}. They proved that for this choice of weights, the scheme detects errors as desired. One difference between the methods is that our new one is all pre-processing while theirs adds extra time to the algorithm as we must compute the checksums using modular arithmetic. Furthermore, the prime that is selected by Jou and Abraham is somewhat larger than the prime we come up with. For example, in [7], if the word length $r = 32$ then the prime $N = 8589934583$. Our scheme is relatively independent of the word length; it depends upon the values of n and d. For $n = 1000$, a large value in light of the dimensions required by current signal processing problems, and $d = 50$, the prime $p = 1051$, which is much smaller than the Jou-Abraham value of N.

We next compare the sizes of the elements in the matrix H for various weight generating schemes. Suppose that $n = 500$ and $d = 10$. Then the range of values for i and j is $i = 1,\ldots,10$ and $j = 1,\ldots,500$. Jou and Abraham proposed the set of weights $w_j^{(i)} = 2^{(i-1)(j-1)}$. While easily implemented as shifts, this set of weights becomes extremely large very quickly, resulting in overflows. For $i = 10$ and $j = 500$, we get $w_j^{(i)} = 2^{4491}$. Another suggestion was to let $w_j^{(i)} = j^{i-1}$[8], which will get large very rapidly, although not as quickly as the previous technique. Using the same example, with $i = 10$ and $j = 500$, we have $w_j^{(i)} = 500^9$. For our new scheme, the smallest prime p satisfying $p>n+d = 510$ is $p = 521$. Therefore, every element $w_j^{(i)}$ will be bounded above by 521. Thus, we see that for this moderately sized problem, our new scheme generates a parity-check matrix whose elements are likewise of moderate size.

6. CONCLUSIONS

In this paper we have proved that, given a consistency check matrix H which defines a distance d+1 code, we can determine the exact number of errors and, if the total number of errors that have occurred lies between 0 and γ, then we can correct all errors. We have also presented a new method for generating the weighted checksum matrix whose elements are bounded in size by the smallest prime p such that $p>n+d$. Since we usually consider $d\ll n$, p is approximately O(n). To generate the matrix H involves only pre-processing steps, and thus will take no additional time in the algorithm.

7. ACKNOWLEDGEMENTS

This work was supported in part by the Joint Services Electronics Program under contract F49620-87-C-0044 at Cornell University. Cynthia Anfinson was also suported by a U.S. Army Science and Technology Graduate Fellowship.

8. REFERENCES

[1] Anfinson, C.J. and Luk, F.T., (1988), "A linear algebraic model of algorithm-based fault tolerance", IEEE Trans. Comput., Vol. C-37, No. 12, pp. 1599-1604.

[2] Brent, R.P. and Luk, F.T., (1985), "The solution of singular-value and symmetric eigenvalue problems on multiprocessor arrays", *SIAM J. Sci. Statist. Comput.*, 6, pp. 69-84.

[3] Brent, R.P., Luk, F.T. and Anfinson, C.J., (1989), "Choosing small weights for multiple error detection", Proc. SPIE, Vol. 1058, High Speed Computing II, pp. 130-136.

[4] Golub, G.H. and Van Loan, C.F., (1983), "Matrix Computations", The Johns Hopkins University Press, Baltimore, Maryland.

[5] Herstein, I.N., (1975), "Topics in Algebra", Second Edition, John Wiley and Sons, New York.

[6] Huang, K.H. and Abraham, J.A., (1984), "Algorithm-based fault tolerance for matrix operations", *IEEE Trans. Comput.*, Vol. C-33, No. 6, pp. 518-528.

[7] Jou, J.Y. and Abraham, J.A., (1986), "Fault-tolerant matrix arithmetic and signal processing on highly concurrent computing structures", Proc. IEEE, Vol. 74, No. 5, Special Issue on Fault Tolerance in VLSI, pp. 732-741.

[8] Luk, F.T., (1985), "Algorithm-based fault tolerance for parallel matrix equation solvers", Proc. SPIE, Vol. 564, Real Time Signal Processing VIII, pp. 49-53.

[9] Luk, F.T. and Park, H., (1988), "An analysis of algorithm-based fault tolerance techniques", *J. Parallel Distrib. Comput.*, Vol. 5, pp. 172-184.

[10] Luk, F.T. and Park, H., (1988), "Fault-tolerant matrix triangulations on systolic arrays", *IEEE Trans. Comput.*, Vol. C-37, No. 11, pp. 1434-1438.

[11] Reed, I.S. and Solomon, G., (1960), "Polynomial codes over certain finite fields", *J. Soc. Ind. Appl. Math.*, Vol. 8, pp. 300-304.

[12] Sarrazin, D.B. and Malek, M., (1984), "Fault-tolerant semiconductor memories", *IEEE Computer, Vol. 17*, No. 8, Special Issue on Fault-Tolerant Computing, pp. 49-56.

[13] Stewart, G.W., (1985), "A Jacobi-like algorithm for computing the Schur decomposition of a non-Hermitian matrix", *SIAM J. Sci. Statist. Comput.*, **6**, pp. 853-864.