CMST 8(1) 79-96 (2002) DOI:10.12921/cmst.2002.08.01.79-96

COMPUTATIONAL METHODS IN SCIENCE AND TECHNOLOGY **8(1)**, 79-96 (2002)

MODIFICATION OF THE WEIGHTED CHECKSUM METHOD FOR DERIVING FAULT TOLERANT VERSIONS OF THE MAIN LINEAR ALGEBRA ALGORITHMS

OLEG MASLENNIKOV

Department of Electronics, Technical University of Koszalin
Partyzantów 17, 75-411 Koszalin, Poland
email: oleg@ie. tu. koszalin.pl

(Received 25 November, 2001)

Abstract. The modified weighted checksum method is proposed, which can be used for deriving fault tolerant versions of most linear algebra algorithms. The purpose is the detection and correction of calculation errors occurred due to transient hardware faults during algorithm execution. Using the proposed method, the fault-tolerant versions of Jordan-Gauss and Faddeeva algorithms are designed. The computational complexity of new algorithms is increased approximately on $O(N^2)$ multiply-add operations in comparison with the original algorithms. However, new algorithms enable to detect and to correct a single error in an arbitrary row or column of input data matrices at the each algorithm step. Hence, it is possible to correct up to N^2 and $(N^2/2 + N \bullet P)$ single errors during realization of whole Jordan-Gauss and Faddeeva algorithms respectively. Finally, the results of experimental verification of the proposed algorithms are represented.

Key words: algorithm-based fault tolerance, weighted checksum method, linear algebra algorithms

1. INTRODUCTION

The methods of linear algebra (LA) make a basis for mathematical models in various fields of science, engineering and technology such as signal and image processing, system theory, statistical and numerical analysis, biomedical researches, physical experiments, etc. For example, here are some specific problems to be solved by modem systems of real-time signal processing: matrix multiplication for covariance estimation, solving of linear systems in adaptive processing, computing eigenvalues/eigenvectors for high-resolution array processing and adaptive beam-forming [1]. However, most of LA algorithms are characterized by a high computational complexity ($O(N^3)$ multiply-add operations, where N is the order of input data matrix) and regularity [1-5]. Therefore, the application-specific parallel systems (ASPS) destined to implementation of several applied algorithms and adapted to their properties are most suitable for real time realization of these algorithms.

Application areas of ASPS demand a large degree of reliability of output results. However, the probability of physical failures increases along with increasing of the algorithm and target computing system complexity. Since a single temporary or permanent failure in a processor can break down an entire computing system, fault tolerance should be provided in these cases on hardware or (and) software levels. The most known methods for providing fault tolerance use hardware or time redundancy, which increase the cost or degrade the performance of compu-

tational systems. Therefore, they are few suitable for real-time computing systems and parallel processors. The algorithm-based fault tolerance (ABFT) methods are more suitable for such systems.

ABFT is an error detection, localization and correction scheme, which uses redundant computations within the algorithms to detect and correct errors caused by transient failures in the hardware, concurrently with normal operation [6-8], In ABFT, the input data are encoded in the form of error detecting or (and) correcting codes. The algorithm is modified to operate on encoded data and produce encoded outputs, from which useful information can be recovered veiy easily. The modified algorithm will be more complex and therefore, will take more time to operate on the encoded data in comparison with the original algorithm. This time overhead should not be excessive. Thus, ABFT methods establish the rules of the original applied algorithms and input data arrays modification. From this, it is clear that these methods are not a general mechanism as some other methods (e.g. the triple modular TMR or triple time redundancy TTR methods), because they may be varied from algorithm to algorithm [18]. However, when the modified algorithm is actually executed on a target architecture, the overheads are required to be minimum in comparison with other known methods. Moreover, in this case, the tolerant to transient fault architectures of ASPS's is derived automatically, using known mapping methods for mapping of fault tolerant algorithms into corresponding PA architectures [3],

Module-level faults are assumed [8] in the algorithm-based fault tolerance. A module (processor or PE for parallel computers) is allowed to produce arbitrary logical errors under physical failure mechanism. This assumption is quite general since it does not assume any technology-dependent fault model. Without loss of generality, a single module error is assumed in this paper. Also, communication links are supposed to be fault-free.

The most known ABFT method called weighted checksum (WCS) one, which is specially tailored to matrix algorithms and array architectures, has been proposed by Abraham et al. [6, 7]. In their scheme, redundancy is encoded at the matrix level by augmenting the original matrix with weighted checksums. Since the checksum property is preserved for various matrix operations, these checksums are able to detect and correct errors in the resultant matrix. Furthermore, the complexity of detection and correction process is much smaller than that of the original computations. For example, the computational complexity of the modified fault-tolerant (FT) version of the matrix multiplication algorithm A(N,N)*B(N,N) = C(N, N) increases on $2N^2$ operations and is equal to $(N^3 + 2N^2)$ multiply-add operations. However, this version allows to detect and to correct the single error among elements of each column of an input matrix A(M, N) occurred during algorithm implementation. Consequently, it enables to correct up to N single errors during solving the whole matrix multiplication task.

However, the original WCS method is not suitable for most LA algorithms, since a single transient fault in a processor or a processor element of an array during computation might cause multiple output errors, which can not be located and corrected. Therefore, in the papers [9-11], we propose the modified WCS method and FT versions of Gauss elimination, Cholesky, Householder reflections and Givens rotations algorithms. In this paper, we establish the sufficient

conditions to use modified WSC-method and then apply its for designing fault-tolerant versions of Jordan-Gauss and Faddeeva algorithms. Finally, the verification results of the proposed algorithms are represented.

2. WEIGHTED CHECKSUM (WCS) METHOD AND ITS MODIFICATION FOR DESIGNING OF FAULT TOLERANT VERSIONS OF MAIN LA ALGORITHMS

The WCS code has been adopted by Jou and Abraham [7] in matrix arithmetic operations for algorithm-based fault tolerance. The idea is to compress the information contained in the row/column elements of matrix into a single element, which is named as a check element. Information is compressed in such a way that it is preserved during algorithm implementation. For example, a WCS encoded data vector **v**(N) with Hamming distance equals three, which can correct a single error (SEC) can be expressed as

$$\mathbf{A_{RC}} = \begin{bmatrix} \mathbf{A} & \mathbf{PRS} & \mathbf{QRS} \\ \mathbf{PCS} & \mathbf{1} & \mathbf{0} \\ \mathbf{QCS} & \mathbf{0} & \mathbf{1} \end{bmatrix} \tag{1}$$

where $\mathbf{v_i}$ is a element of a data vector **v**(N),

$$\mathbf{PCS} = \mathbf{p}^T \mathbf{x}\ [\mathbf{v_1}\ \mathbf{v_2}\ \ldots\ \mathbf{v_N}],$$

$$\mathbf{QCS} = \mathbf{q}^T \mathbf{x}\ [\mathbf{v_1 v_2 \ldots v_N}], \tag{2}$$

and **p**(N), **q**(N) - are encoder vectors.
Possible choices for vector pares **p** and **q** are, for example, [7]
(were **q** is named the exponential weighted encoder vector), or [12]

$$\mathbf{p}^T = [1\ \ 1\ \ldots\ 1] \quad \text{and} \quad \mathbf{q}^T = [2^0\ \ 2^1\ \ \ldots\ \ 2^{N-1}] \tag{3}$$

(were **q** is named the linear weighted encoder vector).

$$\mathbf{p}^T = [1\ \ 1\ \ldots\ 1] \quad \text{and} \quad \mathbf{q}^T = [1\ \ 2\ \ \ldots\ \ N] \tag{4}$$

The difficulty with the first choice is a loss of the numerical accuracy due to large weights, while the second choice leads to larger extra computations necessary to correct an error.

Moreover, the following more advanced encoder vector pares were proposed in the Ref. [13]:

1) average and weighted average encoder vectors:

$$\mathbf{p}^T = [\ 1/N\ 1/N\ \ldots\ 1/N\ ] \text{ and } \mathbf{q}^T = [\ 1/N\ 2/N\ \ldots\ N/N\ ]; \tag{5}$$

2) normalized encoder vectors:

a) for vector a(N)

$$\mathbf{p}^{\mathbf{T}} = [c/|\mathbf{a}|^2 \quad c/|\mathbf{a}|^2 \quad \ldots \quad c/|\mathbf{a}|^2] \tag{6}$$

b) for matrix A(N,N)

$$\mathbf{p}^{\mathbf{T}} = [\ c/|\mathbf{a}|^2 \quad c/|\mathbf{a}|^2 \quad \ldots \quad c/|\mathbf{a}|^2] \text{ and } \mathbf{q}^{\mathbf{T}} = [1 \bullet c/|\mathbf{A}| \quad 2 \bullet c/|\mathbf{A}| \quad \ldots \quad N\text{-}c/|A|], \tag{7}$$

where $|\mathbf{A}|$ is the average value of matrix column (or row) Euclidean norms and c is a constant fixed by user.

Experimental evaluation of numerical error for proposed encoder vectors also were researched in [13] for the set of random generated matrices. The main results are as follows: when round errors are the larger problem, one should use normalized encoder vectors; for overflow problems, one should use average encoder vectors.

Based on the linear encoding vector (4), for example, a matrix **A**(M,N) can be encoded as either a row encoded matrix $\mathbf{A_R}$ given by

$$\mathbf{A_R}=[\mathbf{A} \quad \mathbf{A} \times \mathbf{p}(N) \quad \mathbf{A}\times\mathbf{q}(N)] = [\mathbf{a} \quad \mathbf{PRS} \quad \mathbf{QRS}], \tag{8}$$

where

$$\mathbf{PRS}_i = \mathbf{a}_{i1} + \mathbf{a}_{i2} + \ldots + \mathbf{a}_{iN},$$

$$\mathbf{QRS}_i = 1 \times \mathbf{a}_{i1} + 2 \times \mathbf{a}_{i2} + \ldots + N \times \mathbf{a}_{iN}$$

a column encoded matrix $\mathbf{A_C}$

$$\mathbf{A_C} = \begin{bmatrix} \mathbf{A} \\ \mathbf{p}^{\mathbf{T}}(M)\times\mathbf{A} \\ \mathbf{q}^{\mathbf{T}}(M)\times\mathbf{A} \end{bmatrix} = \begin{bmatrix} \mathbf{A} \\ \mathbf{PCS} \\ \mathbf{QCS} \end{bmatrix}, \tag{9}$$

where

$$\mathbf{PCSj} = \mathbf{a}_{1j} + \mathbf{a}_{2j} + \ldots + \mathbf{a}_{Mj},$$

$$\mathbf{QRSj} = 1 \times \mathbf{ajj} + 2 \times \mathbf{a}_2\mathbf{j} + \ldots + \mathbf{M} \times \mathbf{a}_{Mj},$$

or a full encoded matrix $\mathbf{A_{RC}}$ [12, 14] given by

$$\mathbf{A_{RC}} = \begin{bmatrix} \mathbf{A} & \mathbf{PRS} & \mathbf{QRS} \\ \mathbf{PCS} & 1 & 0 \\ \mathbf{QCS} & 0 & 1 \end{bmatrix}. \tag{10}$$

For example, for matrix multiplication **A**(M,N) * **B**(N,K) = **C**(M,K), the column encoded matrix $\mathbf{A_C}$ of a form (9) is exploited [12]. Then, the following expression is computed:

$$\mathbf{A_C} \times \mathbf{B} = \mathbf{C_C}.$$

To verify the computation, syndromes S_1 and S_2 for the *j*-th column of matrix **C** should be calculated ($j = 1,\ldots, K$):

$$\mathbf{S_1} = \sum_{i=1}^{M} \mathbf{c_{ij}} - \mathbf{PCS_j}$$

and

$$\mathbf{S_2} = \sum_{i=1}^{M} \mathbf{i} \times \mathbf{c_{ij}} - \mathbf{QCS_j}\ .$$

In order to correct a single error, the following procedure (11) is used:

if $\mathbf{S_1 = S_2 = 0}$ then no error has been detected;

if $\mathbf{S_1 \neq 0}$ and $\mathbf{S_2 = 0}$ then PCSj is inconsistent;

if $\mathbf{S_2 \neq 0}$ and $\mathbf{S_1 = 0}$ then QCSj is inconsistent;

if $\mathbf{S_1 \neq 0}$ and $\mathbf{S_2 \neq 0}$ then $\boldsymbol{i} = \mathbf{round(S_2/S_1)}$ and element c_{ij} is erroneous,

and the correction procedure is:

$$\mathbf{c_{ij}} := \mathbf{c_{ij} - S_1}, \tag{11}$$

when $\mathbf{round\ (S_2/S_1)}$ is the nearest integer number to the value $\mathbf{S_2/S_1}$. Note, that the value of $\mathbf{S_2/S_1}$ may be not integer number due to roundoff errors occurred during computations.

Thus, the computational complexity of the modified version of the matrix multiplication algorithm increases only on the $2N^2$ operations and is equal to $(N^3 + 2N^2)$ multiply-add operations. This version allows to detect and to correct the single error among elements of each column of an input matrix A(M,N) occurred during algorithm implementation. Consequently, it is possible to correct up to N single errors during solving the whole task.

However, the original WCS method is not suitable for such LA algorithms as, for example, Gauss elimination, Jordan-Gauss, Faddeeva and Cholesky algorithms, Householder reflections and Givens rotations algorithms, etc., since a single transient fault in a processor or a processor element of an array might cause multiple output errors, which can not be located. In fact, the common property of all above mentioned algorithms is the computation on the any *i*-th algorithm step (may be not one time) the elements of leading (*i*-th) row or/and column of matrix $\mathbf{A^i} = \{a_{ji}{}^{i}\}$ and then modification of other matrix rows (columns) by means of leading ones. The example of corresponding fragment of such algorithms with leading column computations is represented by means construction (12), were values of variables *K*, *K*1, *K*2 and functions *g*1,

$g2$ are depended from the selected algorithm. Note, that in this example the input matrix $\mathbf{A} = \mathbf{A}^1 = \{a_{ji}\}$ is recursively modified during K computation steps to obtain the resulting matrix $\mathbf{A}^{K+1}$.

```
for i:=1 to K do
    begin
{Phase 1:computation of the leading column elements a_ji^(i+1)}
      for j:=i+1 to K1 do
       a_ji^(i+1):=g1(a_ji^i, a_ii^i);                                    (12)
{Phase 2:computation of the elements of the matrix A^(i+1)}
      for j:=i+1 to K1 do
        for k:=i+1 to K2 do
          a_jk^(i+1): = g2(a_jk^i, a_ji^(i+1), a_ik^i);
    end
```

As shown from the (12), if at the i-th algorithm step the element a_{ji}^{i+1} of leading (i-th) column is wrongly calculated, then errors will appear in all elements a_{jk}^{i+1} of j-th row of $\mathbf{A}^{i+1}$. Analogously, if any element a_{ik}^{i} of the leading (i-th) row was wrongly calculated, then errors appear in the all elements of j-th column of $\mathbf{A}^{i+1}$. In both cases, these errors can not be located and corrected by WCS method. If the correction of elements a_{jk} is performed during calculations, then the computational complexity of the original algorithm increases more than twice.

For removing of these defects by means modification of the original WCS method, the following confirmations were proved for all above-mentioned algorithms (see Ref. [9, 10] and the next paper section):

- If during i-th step of computations the element a_{jk}^{i+1} is wrongly calculated, then errors will not appear among others elements of matrix $\mathbf{A}^{i+1}$ while j-th row isn't the leading one (i.e. while $i \neq j$).
- If the element a_{jk}^{i} ($j = i, i + 1, ..., N$) was wrongly calculated several times q ($q < i$) before performing of the i-th step of algorithm (12), then it is possible to correct its using the WCS method for the row encoded matrix A_R (5) at the beginning of the i-th step of the algorithm.
- If an element a_{jk}^{i} ($j = i, i + 1,\ldots,N$) was wrongly calculated during executing of the first phase of the i-th step of algorithm (12), then it is possible to correct it using the WCS method for the column encoded matrix A_C (6) after executing of this phase.

The main consequence of these confirmations is the possibility of performing the detection and correction procedures during each i-th algorithm step among only elements of the leading (i-th) row and leading (i-th) column of the matrix $\mathbf{A}^i$. Based on these confirmations, the modification of the origin WCS method was performed. The main idea of the proposed unified WCS method (scheme) destined for main linear algebra algorithms is the performing of its check procedures concurrently with algorithm computations or more exactly, the performing of the detection and correction procedures:

- at each i-th algorithm step;
- among only elements of the leading (i-th) row and leading (i-th) column of matrix A^i.

Note, that proposed modified method may be used to design the fault-tolerant version of an arbitrary matrix algorithm for which above-mentioned confirmations are corrected. Therefore, these confirmations may be considered as sufficient conditions for using the modified WCS method.

As a result, the proposed checksum scheme increases the computational complexity of original algorithm (12) approximately on $0(N^2)$ operations (such as multiply-add operations). Consequently, the proposed modification of WCS-method does not increase its computational complexity. However, the proposed uniform scheme enables to correct one error among elements of an arbitrary column (or row) of an input matrix A(M, N) on any from K steps of algorithm implementation. Consequently, it is possible to correct up to K (where K = (N - 1) for case M = N) errors during solving the whole LA task.

In the next section of this paper, we will try to use the proposed modified WCS method to design the fault tolerant version of Faddeeva and Jordan-Gauss algorithms.

3. DESIGN OF THE FAULT TOLERANT VERSION OF FADDEEVA AND JORDAN-GAUSS ALGORITHMS

Starting with N x N, Nx K, P x N and P X R input matrices **A, B, C** and **D,** respectively, Faddeeva algorithm is intended [2, 5, 15] for solving matrix equations of the type

$$X=C\cdot A^{-1}\cdot B+D \tag{13}$$

were the four input matrices form an (N + P) x (N + R) joint matrix $\tilde{F}$ when arranged in the following way:

$$\tilde{F} = \begin{bmatrix} A & B \\ -C & D \end{bmatrix} \tag{14}$$

The idea of Faddeeva algorithm consists of reducing the lower left quadrant of the matrix $\tilde{F}$ (i.e. **C**-matrix) to zero matrix, while in the lower right quadrant of the matrix $\tilde{F}$ is formed to the resultant PxR matrix **X**. In order to perform above-stated operations with **A** being a non-singular matrix, the Gauss elimination algorithm is used. Hence, in the course of computations, the joint matrix $\tilde{F}$ is being transformed into the following matrix:

$$F' = \tilde{F}^{n+1} = \begin{bmatrix} U & B^* \\ O & X \end{bmatrix} \tag{15}$$

where **U** is the upper triangular matrix.

The main practical advantage of Faddeeva algorithm is its versatility. This stems from the fact that expression (13) allows to solve a set of problems. Some of them are listed below:

- solving a system $AX = B$ of linear algebraic equations with one or more right-hand sides (depending upon the numbers of columns in B), i.e.

$$X = A^{-1}B \quad \text{for} \quad C = I, \quad D = 0$$

where I is the identity matrix;
- matrix multiplication $X = C \cdot B$ for $A = I$, $D = 0$;
- matrix multiply-add operation $X = C \cdot B + D$ for $A = I$;
- matrix inversion $X = A^{-1}$ for $C = B = I$, $D = 0$;
- adaptive filtering algorithms $X = C \cdot A^{-1} + D$ for $B = I$.

There are other important modifications of Faddeeva algorithm. It can be employed, for example, in fast solving of linear programming problems using Karmarkar algorithm.

To provide a numerical stability of Faddeeva algorithm, Gauss elimination with partial pivoting within columns [2, 16, 17] is usually used. As a result, at the i-th step ($i = 1,..., N$) of the algorithm, the elimination of elements $f_{ji}^{i}(j = i + 1, ..., N + P)$, which belong either to the original matrix $\tilde{F} = \tilde{F}^{1}$ (for $i = 1$) or to the partially transformed matrix $\tilde{F}^{i}$ (for $i > 1$), is preceded by successive comparisons of f_{ji}^{i} ($j = i + 1,..., N$) with the pivot element f_{ji}^{i}. If

$$|f_{ji}^{i}| > |f_{ii}^{i}|,$$

then the i-th and j-th rows of the matrix $\tilde{F}^{i}$ are interchanged and a Boolean variable v_{ji} is set to 1. In the opposite case, the row interchange doesn't take place, and v_{ji} is set to 0. After completing all comparisons and interchanges for a given step, the pivoting (i-th) row with the pivoting element $f_{ii}^{i}$is finally derived. Then the original Gauss elimination of the elements f_{ji}^{i} ($j = i + 1,..., N + P$) starts. It is accompanied by calculations of elements m_{ji} of the lower triangular matrix M and transformations of rows of the matrix $\tilde{F}$, from the ($i + 1$) row to the ($N + P$) row.

However, to provide a correct realization of the algorithm, the selection of pivoting elements as well as corresponding interchanges are limited only to the upper (corresponding to the matrices A and B) quadrants of matrices $\tilde{F}^{i}$. Note, that the elimination process is carried out within all quadrants of $\tilde{F}^{i}$. Naturally, in the N-th step, the element $f_{N}^{N}{}_{N}$ is immediately taken as a pivoting one, without any comparison. The described above version of Faddeeva algorithm can be expressed by the following Pascal-like form:

```
for i:=1 to N do
begin
   for j:=i+1 to N do
    begin
      {pivoting}
       if abs(fii) < abs(fji)
        then begin s:=fii; fii:=fji; fji:=s; vji:=1; end
        else vji:=0;
      {row interchanges}
```

```
      for k:=i+1 to N+R do
       if v_ji:=1 then begin temp:=f_ik; f_ik:=f_jk; f_jk:=temp; end;
   end {j};
      {calculation of the elements m_ji}
        for j:=i+1 to N+P do                                        (16)
           m_ji:=-f_ji / f_ii;
      {elimination}
        for j:=i+1 to N+P do
        for k:=i+1 to N+R do
          f_jk:=f_jk+m_ji*f_ik;
   end;
```

The Jordan-Gauss algorithm [16,17] is an efficient alternative to classical Gauss elimination for the solution of dense linear systems of the form

$$A \cdot \bar{x} = \bar{b} \tag{17}$$

where **A** is N x N matrix of the system coefficients. The main advantage is that it gathers together two phases, triangularisation and back substitution [17]. In the case when **X** and **B** are N x R matrices, this algorithm is the particular case of Faddeeva algorithm, in which the two input matrices **A** and **B** form an joint (N + N) x (N + R) matrix $\tilde{F}$ of the following form:

$$\tilde{F} = \begin{bmatrix} A & B \\ -I & 0 \end{bmatrix}, \tag{18}$$

where I is the identity matrix, and 0 is a zero matrix. Then the N steps of Gauss elimination are performed for transforming the matrix $\tilde{F}$ into the matrix F' (15).

To provide numerical stability of this algorithm, the Gauss elimination with partial pivoting may be used. The described above version of Jordan-Gauss algorithm can be expressed in the following form:

```
for i:=1 to N do
   begin
   {selection of the pivoting element}
   for j:=i+1 to N  do
      begin
        if abs(f_ii^i)<abs(f_ji^i) then begin s:=f_ii^i; f_ii^i:=f_ji^i; f_ji^i:=s; v_ji=1; end
                                   else v_ji:=0;
   {row interchanges}
      for k:=i+1 to N+R do
        if v_ji=1 then begin s:=f_ik^i; f_ik^i:=f_jk^i; f_jk^i:=s; end;
   end {j};
   {calculation of elements m_ji}
        for j:=i+1 to N+i do                                        (19)
          m_ji:=-f_ji^i/f_ii^i;
   {elimination}
      for j:=i+1 to N+i do
```

```
        for k:=i+1 to N+R do
          f_jk^(i+1) := f_jk^i + m_ji * f_ik^i;
end;
```

where

$$f_{j\,k}^{1} := a_{j\,k}, \qquad j = 1, 2, ..., N, \qquad k = 1, 2, ..., N;$$

$$f_{j(N+p)}^{1} := b_{jp}, \qquad j = 1, 2, ..., N, \qquad p = 1, 2, ..., R;$$

$$f_{(N+1)i}^{1} := -1, \qquad i = 1, 2, ..., N;$$

$$f_{(N+i)l}^{1} := 0, \qquad i = 1, 2, ..., N, \qquad l = i+1, i+2, ..., N+R.$$

As a result of the execution of this algorithm, the desired elements of the matrix **X** are determined as follows:

$$x_{jp} = f_{(N+j),(N+P)}^{(N+1)}, \quad j = 1, 2, \ldots N, \; p = 1, 2, \ldots K \tag{20}$$

It is followed from the constructions (16) and (19), that if during i-th computation step the element m_{ji} is wrongly calculated, then errors will appear in all elements f_{jk}^{i+1} of j-th row of $\tilde{F}^{i+1}$. Moreover, if any element f_{ik}^{i} of the leading row is wrongly calculated, then errors appear in all elements of k-th column of $\tilde{F}^{i+1}$. In both cases, these errors can not be corrected by the original WCS-method. Therefore, in order to derive a fault tolerant version of this algorithm, the proposed modified WCS method should be used. However, the conditions represented in the previous section should be true. For algorithms (16) and (19) these conditions are transformed in the theorems 1,2 and 3 respectively.

Theorem 1

If during the i-th step of the algorithms (16) or (19) the element f_{jk}^{i+1} was wrongly calculated, then errors do not appear among other elements of matrix $\tilde{F}^{i+1}$, while the j-th row isn't the pivoting one (i.e. $i \neq j$).

The proof of this theorem directly follows from the algorithm (16), where each element f_{jk}^{i} takes part in calculations only elements f_{jk}^{i+1}, f_{jk}^{i+2},..., f_{jk}^{i+g}, where $(i+g) \le j$ and $(i+g) \le k$.

Theorem 2

Assume that the element f_{jk}^{i} was wrongly calculated q times ($q < i$) before executing the i-th step of algorithm (16) or (19). Then it is possible to correct its value only once, using WCS method for row encoded matrix $\tilde{F}^{i+1}$, at the beginning of the i-th step of the corresponding algorithm.

Proof

Without the loss of a generality, we assume that $i < j$, $i < k$ and $q = 2$ for element f_{jk}^{i}. Let the element f_{jk}^{i} was wrongly calculated at the $(i-1)$-th step of algorithms (16) or (19). Then its value will be equal to

$$f_{jk}^{iz} = f_{jk}^{i} + z_{jk}^{i},$$

where z_{jk}^{i} - is the calculation error. Then, in accordance with (3.16), after performing of next algorithm step, we obtain:

$$f_{ik}^{i+1} = f_{jk}^{iz} + m_{ji} \cdot f_{ik}^{i} = f_{jk}^{i} + z_{jk}^{i} + m_{ji} \cdot f_{ik}^{i}.$$

We assume now, that last expression was also wrongly calculated. In similar way, we obtain that the value of the element f_{jk}^{i+1} will be equal to

$$f_{jk}^{(i+1)z} = f_{jk}^{(i+1)} + z_{jk}^{i+1} = f_{jk}^{i} + z_{jk}^{i} + z_{jk}^{i+1} + m_{ji} \cdot f_{ik}^{i} = f_{jk}^{i} + z_{jk} + m_{ji} \cdot f_{ik}^{i},$$

where $z_{jk} = z_{jk}^{i} + z_{jk}^{i+1}$.

Thus, the computation errors of the element f_{jk} are accumulated in the variable z_{jk}. Consequently, the wrongly calculated element f_{jk} may be corrected only at the beginning of the j-th step of the algorithm (16) or (19), i.e. when the j-th row will become the leading row ($j=i$).

Theorem 3

Values of the checksum CS_i and the weighted checksum WCS_i of the i-th column of the matrix M are respectively equal to values of the checksum $PCS_i^{(i+1)}$ and the weighted checksum $QCS_i^{(i+1)}$ of the i-th column of matrix $\tilde{F}^{i+1}$, i.e. equals to values of the checksum and the weighted checksum of i-th column of matrix $\tilde{F}$ after performing the i-th step of the algorithms (16) or (19).

Proof

At the beginning of the i-th step of algorithm (16) the values PCS_i^i and QCS_i^i of the matrix F_C in accordance to (9) are equal to the following expressions:

$$PCS_i^i = f_{ii}^{i} + f_{(i+1)i}^{i} + \ldots + f_{(N+P)i}^{i}$$

and

$$QCS_i^i = i \cdot f_{ii}^{i} + (i+1) \cdot f_{(i+1)i}^{i} + \ldots + (N+P) \cdot f_{(N+P)i}^{i}$$

respectively.

After performing of the i-th step of the algorithm (3.16) with the column encoded matrix F_C, these values will be equal to

$$PCS_i^{(i+1)} = PCS_i^i / f_{ii}^{i} \quad \text{and} \quad QCS_i^{(i+1)} = QCS_i^i / f_{ii}^{i}.$$

In other side, values of the checksum CS_i and the weighted checksum WCS_i of the i-th column of the matrix M in accordance to the expression (9) and algorithm (16) are equal to following expressions:

$$CS_i = 1 + m_{(i+1)i} + m_{(i+2)i} + \ldots + m_{(N+P)i} =$$
$$= 1 + f^{i}_{(i+1)i} / f^{i}_{ii} + \ldots + f^{i}_{Nk} / f^{i}_{ii} = PCS^{i}_{i} / f^{i}_{ii}$$

and

$$WCS_i = i + (i+1) \cdot m_{(i+1)i} + (i+2) \cdot m_{(i+2)i} + \ldots + (N+P) \cdot m_{(N+P)i} =$$
$$= i + (i+1) \cdot f^{i}_{(i+1)i} / f^{i}_{ii} + \ldots + (N+P) \cdot f^{i}_{(N+P)k} / f^{i}_{ii} = QCS^{i}_{i} / f^{i}_{ii}.$$

Note that the proof of this theorem for the algorithm (19) performs in a similar way. Thus, the correctness of conditions represented in the section 2 is proved. Therefore, in order to derive of a fault tolerant version of this algorithm, the proposed modified WCS method checksum scheme may be used.

However, we should be certain that the elements of *i*-th column of matrix *F'* were calculated correctly at the (*i* - 1) step of the algorithms (16) and (19). It is easy proved that correctness of these elements may be verified using WCS-scheme for *i*-th column of matrix $F_C^{\,i}$ (analogously to proof of the theorem 3). Finally, the fault tolerant version of Faddeeva and Jordan-Gauss algorithms without pivoting consists of execution of the following stages:

1. The original matrix **F** is represented in the form of the fully encoded matrix $\mathbf{F}_{RC}$ (see expression (10))

$$\mathbf{F}_{RC} = \begin{bmatrix} \mathbf{f}_{11} & \mathbf{f}_{12} & \cdots & \mathbf{f}_{1,N+R} & \mathbf{PRS}_1 & \mathbf{QRS}_1 \\ \mathbf{f}_{21} & \mathbf{f}_{22} & \cdots & \mathbf{f}_{2,N+R} & \mathbf{PRS}_2 & \mathbf{QRS}_2 \\ \vdots & \vdots & & \vdots & \vdots & \vdots \\ \mathbf{f}_{N+P,1} & \mathbf{f}_{N+P,2} & \cdots & \mathbf{f}_{N+P,N+R} & \mathbf{PRS}_{N+P} & \mathbf{QRS}_{N+P} \\ \mathbf{PCS}_1 & \mathbf{PCS}_2 & \cdots & \mathbf{PCS}_{N+R} & \mathbf{1} & \mathbf{0} \\ \mathbf{QCS}_1 & \mathbf{QCS}_2 & \cdots & \mathbf{QCS}_{N+R} & \mathbf{0} & \mathbf{1} \end{bmatrix}$$

where values of the checksums and weighted checksums are represented by the following expressions (in the case of using the linear encoded vector (4)):

$$\begin{aligned} \mathbf{f[i, N+R+1] = PRS_i = f_{i1} + f_{i2} + \ldots + f_{i,N+R},} \\ \mathbf{f[i, N+R+2] = QRS_i = 1 \times f_{i1} + 2 \times f_{i2} + \ldots + (N+R) \times f_{i,N+R},} \\ \mathbf{f[N+P+1, j] = PCS_j = f_{1j} + f_{2j} + \ldots + f_{N+P,j},} \\ \mathbf{f[N+P+2, j] = QCS_j = 1 \times f_{1j} + 2 \times f_{2j} + \ldots + (N+P) \times f_{N+P,j}.} \end{aligned} \tag{21}$$

2. For *i* = 1, 2,..., *N* - 1, stages 3-7 are repeated.

3. At the beginning of the *i*-th algorithm step, error detection and correction procedure within elements belong to the *i*-th column row of the matrix $\tilde{F}^{\,i}_{RC}$ is performed in accordance to the expressions (11).

4. The error detection and correction procedure within elements which belong to the *i*-th row of $\tilde{F}^{i}_{RC}$ matrix is performed in accordance to the procedure (11).

5. The elements m_{ji} are calculated.

6. The error detection and correction procedure for the elements m_{ji} is performed in accordance to the procedure (11).

7. The elements of matrix $\tilde{F}^{i+1}_{RC}$ are calculated.

Note, that realization of the detection and correction procedures for elements of *i*-th column of matrix $\tilde{F}^{i}$ and elements m_{ji} requires to perform 2*N multiply-add operations and 2*N additions in the case of Jordan-Gauss algorithm, and 2*(N - i + P) multiply-add operations and 2*(N - i + P) additions in the case of Faddeeva algorithm. For realization of the detection and correction procedure for the elements of *i*-th (leading) row of the matrix $\tilde{F}^{i}$ it is necessary to perform (N - i + R) operations of multiplication with addition and (N - i + R) operations of addition for both Jordan-Gauss and Faddeeva algorithms. Moreover, for both mentioned algorithms, the resulting elements x_{jp} are not correct during computations. Therefore, these elements should be checked and corrected after algorithm implementation by means the original checking procedure of the WCS method. For realization of this stage, it is necessary to perform N*R operations of multiplication with addition and N*R operations of addition. This means, that the computational complexity of the whole FT Faddeeva and Jordan-Gauss algorithms increases on $0(N^2)$ multiply-add operations and $0(N^2)$ additions. Besides, due to increasing input matrix sizes, the computational complexity of the proposed algorithms is also increased on $O(N^2)$ multiply-add operations (see Table 1) in comparison with the original algorithms. However, new algorithms enable to detect and to correct one error in an arbitrary row or column of the matrix $\tilde{F}^{i}$. This means, that it is possible to correct up to N errors during whole of Jordan-Gauss or Faddeeva algorithms.

Table 1. Computational complexity of the proposed FT algorithms

No	Algorithm	Computational complexity of the algorithm		Numbers of detected and corrected errors	
		Original	Fault-tolerant	within row (column)	For whole algorithm
1	Gauss elimination	$N^3/3$	$N^3/3 + 3.5 \cdot N^2$	1	N
			$N^3/3 + 6.5 \cdot N^2$	1 (each step)	$N^2/2$
2	Jordan-Gauss	$N^3/2 + N^2 \cdot R$	$N^3/2 + N^2R + 2.5N^2 + 2N \cdot R$	1	N + R
			$N^3/2 + N^2R + 5N^2 + 3N \cdot R$	1 (each step)	$N^2 + R$
3	Faddeeva	$N^3/3 + N \cdot P \cdot R + + N^2(R + P)/2$	$N^3/3 + N \cdot P \cdot R + N^2(R + P)/2 + + RP + 1.5N^2 + N \cdot (R + 2P)$	1	N + R
			$N^3/3 + N \cdot P \cdot R + N^2(R + P)/2 + + RP + 3N^2 + 2N \cdot (R + 2P)$	1 (each step)	$N^2/2 + R$

In the Pascal-like form the fault tolerant version of Faddeeva algorithm without pivoting may be represented by the following construction (22), were δ is a small value, named a tolerance, so that a row (column) of resulting matrix will be accepted as error-free if the difference between the computed row (column) sum and checksum is less than δ. The variable ε is a machine depended constant with a small (roundoff) value. Note, that in the case when a single error was occurred during computations, the difference (**round** ($S2/S1$) - $S2/S1$) is determined by only roundoff values and therefore has a very small value. At the same time, if more errors were occurred within one column (row) computations, this difference has not a small value. Therefore, the variable ε is used here for searching of multiply errors and halting program execution.

```
for i:= 1 to N do
 begin
{Error detection and correction within elements of the i-th column of F~^i}
  PCS_i:=0; QCS_i:=0;
  for j:=i to N+P do begin PCS_i:= PCS_i+f_ji; QCS_i:=QCS_i+j*f_ji; end;
  S1:=PCS_i-f_N+P+1,i; S2:=QCS_i-f_N+P+2,i;
  if abs(S1)>δ and abs(S2)<δ then f_N+P+1,i:=PCS_i;
  if abs(S2)>δ and abs(S1)<δ then f_N+P+2,i:=QCS_i;
  if abs(S1)>δ and abs(S2)>δ then
      if abs(round(S2/S1)-S2/S1)<ε then begin j:= round(S2/S1); f_ji:=f_ji-S1; end
                                   else halt;
{Error detection and correction within elements of the i-th row of F~^i}
  PRS_i:=0; QRS_i:=0;
  for k:=i to N+R do begin PRS_i:=PRS_i+f_ji; QRS_i:=QRS_i+k*f_ji; end;
  S1:=PRS_i-f_i,N+R+1; S2:=QRS_i-f_i,N+R+2;
  if abs(S1)>δ and abs(S2)<δ then f_i,N+R+1:= PRS_i;
  if abs(S2)>δ and abs(S1)<δ then f_i,N+R+2:=QRS_i;
  if abs(S1)>δ and abs(S2)>δ then
      if abs(round(S2/S1)-S2/S1)<ε then begin k:=round(S2/S1) f_ik:=f_ik-S1; end
                                   else halt;
{Computation of elements m_ji of the matrix M}
  for j:=i+1 to N+P+2 do
   m_ji:=-f_ji/f_ii ;
{Error detection and correction within elements of the i-th column of matrix M}
  CS_i:=1; WCS_i:=i;
  for j:=i+1 to N+P do begin CS_i:= CS_i+m_ji; WCS_i:= WCS_i+j*m_ji; end;          (22)
  S1:=CS_i-m_N+P+1,i; S2:=WCS_i-m_N+P+2,i;
  if abs(S1)>δ and abs(S2)<δ then m_N+P+1,i:=CS_i;
  if abs(S2)>δ and abs(S1)<δ then m_N+P+2,i:=WCS_i;
  if abs(S1)>δ and abs(S2)>δ then
      if abs(round(S2/S1)-S2/S1)<ε then begin j:= round (S2/S1); m_ji:=m_ji-S1; end
                                   else halt;
{Modification of elements of the matrix F~^i}
  for j:=i+1 to N+P+2 do
    for k:=i+1 to N+R+2 do
      f_jk:=f_jk+m_ji*f_ik;
end;
```

where

$$f_{jl}^{1} = a_{jl}, \quad j = 1, 2, \ldots, N, \quad l = 1, 2, \ldots, N;$$
$$f_{j(N+k)}^{1} = b_{jk}, \quad j = 1, 2, \ldots, N, \quad k = 1, 2, \ldots, R;$$
$$f_{(N+p)l}^{1} = -c_{pl}, \quad p = 1, 2, \ldots, P \quad l = 1, 2, \ldots, N;$$
$$f_{(N+p)k}^{1} = d_{pk}, \quad p = 1, 2, \ldots, P, \quad k = 1, 2, \ldots, R.$$

Note that the Pascal-like form of the FT Jordan-Gauss algorithm without pivoting may be also represented by the construction (22), in which the variable *i* is used instead of the parameter *P*.

It is assumed in the construction (22), that no error occurs during calculation of the checksum and weighted checksums PCS_i, QCS_i, CS_i and WCS_i, i.e. these values should be calculated by fault-tolerant hardware of a system. In the opposite case, the fault-tolerant version of an algorithm must provide the double recomputation of the erroneous checksum or weighted checksum value in accordance to the following construction (23):

```
PCS_i:=0; QCS_i:=0;
  for j:=i to N do begin PCS_i:= PCS_i+a_ji; QCS_i:=QCS_i+j*a_ji; end;
  S1:=PCS_i-a_N+1,i; S2:=QCS_i-a_N+2,i;
  if abs(S1)>δ and abs(S2)<δ then begin
      for s:=1 to 2 do begin b(s):=0; for j:=i to N do b(s):=b(s)+a_ji; end;
       if PCS_i=b(1) or PCS_i=b(2) then a_N+1,i:=PCS_i else a_N+1,i:=b(1); end;
  if abs(S2)>δ and abs(S1)<δ then begin                                    (23)
      for s:=1 to 2 do begin b(s):=0; for j:=i to N do b(s):= b(s)+j*a_ji; end;
       if QCS_i=b(1) or QCS_i=b(2) then a_N+2,i:=QCS_i else a_N + 2,i:=b(1); end;
  if abs (S1)>δ and abs(S2)>δ then
      if abs(round(S2/S1)-S2/S1)<ε then begin j:=round(S2/S1);a_ji:=a_ji-S1; end
                                   else halt;
```

Note, that in the case when the construction (23) is used in the proposed algorithms, the computational complexity of whole FT Faddeeva and Jordan-Gauss algorithms do not increase (in comparison to the construction (22)) when no errors is occurred in the checksums or weighted checksums. However, $2 \cdot (N - i)$ extra multiply-add operations for each error, which was occurred in the checksums or weighted checksums at the *i*-th algorithm step arc needed. However, such versions of FT algorithms enable to detect and to correct a single error in an arbitrary row or column of the matrix $\tilde{F}^{i}$ at each algorithm step. This means, that it is possible to correct up to N^2 and $N^2 + N \cdot P$ errors during whole of FT Jordan-Gauss and Faddeeva algorithms implementation respectively. The computational complexity of the proposed algorithms is represented in the Table 1.

Remark

Faddeeva (16) and Jordan-Gauss (19) algorithms with partial pivoting differ from the original ones only in the extra procedures for leading row selection. Because the procedures of elements comparing and row interchanges don't influence on the checksum and weighted checksum values, the theorems 1-3 and all stages of the fault tolerant versions of the mentioned algorithms without

pivoting will also true when the strategy of partial pivoting is used. However, for partial pivoting version, after performing the stage 3, the selection of the pivoting element and corresponding row exchanges should be executed first. Then the stage 4 of the proposed algorithms may be carried out.

4. EXPERIMENTAL VERIFICATION OF THE PROPOSED FT ALGORITHMS

In order to estimate a tolerance of the proposed algorithms to transient faults (i.e. calculation errors) and for evaluation of the numerical error for different matrix sizes and types and different encoder vectors, the program environment "ABFT" was designed in Borland Delphi environment. This program allows:

- to control the process of execution fault tolerant algorithms for different input data and checksum types: Single, Real, Double;
- to select the type of the encoded vector: Linear or Average;
- to select the numerical accuracy of computations;
- to select the breakpoints: step over or after error finding;
- to inject the single errors and to select the error values: for each algorithm, in an arbitrary algorithm step, into an arbitrary matrix element;
- to type the results of the algorithm implementation and detected errors;
- to read of input data from files and to write of the results to output files;

The main testing results of the proposed algorithms with different encoder vectors and input matrices are following (see also Table 2):

- the proposed fault tolerant versions of Faddeeva and Jordan-Gauss algorithms permit to detect and to correct a single error in each column of input matrix at each algorithm step;
- the advantage of use of the average encoder vector in comparison with the linear one is that it will not cause any overflow error unless there is an overflow error in the computation of the information matrix [13]. However, the drawback of these encoder vectors as well as the linear checksum encoder vectors is that they can be used only for well conditional input matrices. Moreover, the average encoder vector can be used only for matrices, which elements have values of approximately the same order as the size N of input matrices. An experimental evaluation shows that even for matrices with ratio S of the average value E of elements to the matrix size N less then $|S| < 0.05$, the linear encoder vector is better in comparison with the average one (i.e. allows to detect and to correct the smaller error). Thus, in a case of using both mentioned encoder vectors, the minimal detected and corrected error depends on values of the input matrix elements. Therefore, the best choice is the normalized encoder vector, which allows automatically to adapt the checksum coefficients to values of the input matrix elements. Mathematical details of numerical properties of the proposed encoder vectors are not the object of this paper and may be a base of separate investigations.
- for different input matrix elements and checksum data types (Single, Real or Double), i.e. for different input data precision, the minimal error value or the value of a tolerance δ in

the proposed algorithms must be equal the data precision value for small size well conditioned ($cond < 100$) input matrices and more value for larger matrices and/or matrices with a higher condition number. For example, for input data of Real format (accuracy is equal 10E-11), input matrix size equal 40 and condition number $cond = 134$, the error $\delta \geq$ 10E-10 may be detected and corrected in the case of Jordan-Gauss algorithm with partial pivoting and with the linear weighted encoded vector. Thus, the increasing both input matrix sizes and/or *cond* value require an increase of the tolerance value δ.

Table 2. Experimental verification of proposed FT Jordan-Gauss algorithm

Matrix sizes and conditional number	Data type	CS and WCS type	Linear encoded vector		Average encoded vector	
			Precision	Minimal error δ	Precision	Minimal error δ
20 × 20, $cond = 19$	Single	Real	10E-7	10E-6	10E-7	10E-6
	Real	Real	10E-7	10E-7	10E-7	10E-7
	Real	Real	10E-11	10E-11	10E-11	10E-10
	Double	Double	10E-15	10E-15	10E-15	10E-14
40 × 40, $cond = 134$	Real	Real	10E-11	10E-10	10E-11	10E-9
	Double	Double	10E-15	10E-14	10E-15	10E-13
100 × 100, $cond = 491$	Real	Real	10E-7	10E-5	10E-7	10E-5
	Real	Double	10E-8	10E-7	10E-8	10E-6
	Real	Double	10E-12	10E-10	10E-12	10E-9
100 × 100, $cond = 932$	Real	Real	10E-7	10E-5	10E-7	10E-4
	Real	Double	10E-8	10E-6	10E-7	10E-5
	Real	Double	10E-11	10E-9	10E-11	10E-8

5. CONCLUSIONS

The modification of the original WCS method, which operates with a wider set of LA algorithms and allows to design the effective fault-tolerant versions of algorithms has been described, as well as the sufficient conditions for using of the proposed method have been formed. The fault tolerant versions of Faddeeva and Jordan-Gauss algorithms were designed using modified WCS method. The computational complexity of new algorithms increase approximately on $0(N^2)$ multiply-add operations in comparison with the original algorithms. However, new algorithms enable to detect and to correct a single error in an arbitrary row or column of the input matrix at the each algorithm step. Hence, it is possible to correct up to N^2 and $(N^2/2 + N \cdot P)$ single errors during realization of whole Jordan-Gauss and Faddeeva algorithms respectively (see Table 1). The verification of the proposed algorithms proved that they are correct and they enable to detect and to correct a single error in an arbitrary row or column of

the well conditioned input matrix at the each algorithm step. It has been established, that the drawback of using the average and the linear encoder vectors is that the minimal detected and corrected error depends on values of the input matrix elements. Therefore, the best choice is the use of the normalized encoder vector, which allows automatically to adapt the checksum coefficients to values of the input matrix elements. Moreover, the minimal error value or the value of a tolerance δ in the proposed algorithms can be equal the data precision value for small size well conditioned ($cond < 100$) input matrices and must be more value for larger matrices and/or matrices with a higher conditional number.

References

[1] S. Y. Kung, H. J. Whitehouse, T. Kailath, *VLSI and Modern Signal Processing,* Prentice-Hall, Englewood Cliffs, New Jersey (1988).

[2] G. H. Golub, C. F. V. Loan, *Matrix Computations,* Baltimore: John Hopkins Univ. Press (1983).

[3] S. Y. Kung, *VLSI Array Processors.* Englewood Cliffs, N.J. Prentice Hall (1988).

[4] M. Cosnard, D. Trystram, *Parallel Algorithms and Architectures,* International Thomson Computer Press, Boston (1995).

[5] D. K. Faddeev, V. N. Faddeeva, *Computational methods of linear algebra,* W. H. Freeman and Company (1963).

[6] K. H. Huang, J. A. Abraham, *Algorithm-based fault tolerance for matrix operations,* IEEE Trans. Comput., **C-33,** 518 (1984).

[7] J. Y. Jou, J. A. Abraham, *Fault-tolerant matrix arithmetic and signal processing on highly concurrent computing structures,* Proc. IEEE, **5(74)**, 732 (1986).

[8] L. Hammond, B. Nayfeh, K. Olukotun, *A Single-Chip Multiprocessor,* Computer, **30(9),** 79 (1997)

[9] R. Wyrzykowski, J. Kanevski, N. Maslennikova, O. Maslennikov, *Fault-Tolerant Matrix Decomposition and its Implementation on Processor Arrays,* Engineering Simulation, **15(6)**, 779 (1998), Gordon&Breach Science Publishers, England.

[10] O. Maslennikov, A. Guzinski, J. Kanevski, R. Wyrzykowski, *Fault tolerant QR-Decomposition Algorithm Based on Householder Reflections and its Parallel Implementation*, Proc.4-th Int. Workshop Parallel Numerics'97, Zakopane (Poland), 177 (1997).

[11] O. Maslennikov, J. Kanevski, R. Wyrzykowski, *Fault tolerant QR-decomposition algorithm and its parallel implementation,* Lecture Notes in Computer Science, D. Pritchardand, J. Reeve (Eds.), Springer, 1470, 798 (1998).

[12] D. E. Schimmel, F. T. Luk, *A practical real time SVD machine with multi-level faidt tolerance,* SPIE Real time signal processing IX, **698,** 142 (1986).

[13] V. S. S. Nair, J. A. Abraham, *Real-number codes for faidt-tolerant matrix operations on processor arrays*, IEEE Trans.on Comp., **39(4),** 426 (1990).

[14] F. T. Luk, H. Park, *An analysis of algorithm- based tolerance techniques,* SPIE Vol.696, Advanced algorithms and architectures for signal processing, pp. 222-227 (1986).

[15] R. Wyrzykowski, J. Kanevski, O. Maslennikov, *Systolic-type implementation of matrix computations based on the Faddeeva algorithm,* Proc. IEEE Int. Conf. Massively Parallel Computing Systems, Ischia (Italy), pp. 31-42 (1994).

[16] A. Jennings, J. J. McKeown, *Matrix computations,* Willey&Sons, Chichester (1992).

[17] J. M. Ortega, *Introduction to parallel and vector solution of linear systems,* Plenum Press, New York (1988).

[18] M. Vijay, R. Mittal, *Algorithm-based fault tolerance: a review,* Microprocessors and Microsystems, **21,** 151 (1997).