Fault Tolerant QR - decomposition Algorithm Based on Householder’s (Reflections) Method and its Parallel Realisation.

Workpackage WP5.8

O. Maslennikov, D. Chvyrov, A. Guzinski and Ju. Kanevski

Electronic Institute, Technical University of Koszalin
Partyzantow 17, 75-411 Koszalin, Poland
E-mail: oleg@kos.man.koszalin.pl
Abstract.

A fault-tolerant QR-decomposition algorithm based on Householder’s method of reflections and modified weighted checksum method is proposed. The purpose is to detect, locate and correct the calculation errors occurred due to transient hardware faults during computation. The proposed algorithm enables to correct a single error among elements of any column (or row) of an input matrix $A(M,N)$ on any from $N$ steps of algorithm implementation. Consequently, it is possible to correct up to $N$ single errors during solving the whole decomposition task. This effect is obtained by increasing the computational complexity of the original Householder’s method on $8.5N^2 + O(N)$ multiply-add operations. Finally, the parallel version of proposed algorithm destined to realization on a fixed-size linear processor array architecture with fully local communications and low I/O requirements is designed.

1 Introduction.

Many applied problems from various fields of science and technology are reduced to implementations of linear algebraic problems with different kinds dense matrix. However, most of these problems are characterised by high computational complexity [1, 2] (O($N^3$) multiplication with addition operations, where $N$ is the order of input matrix $A$). This implies the necessity of solving these problems on high performance consequential or parallel computers (and, in particular, on VLSI processor arrays [3 - 5]). The application areas of these computers demand a large degree of reliability of output results. However, along with the increasing of computational complexity and complexity of computers, the probability of physical failures increases. 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 or on hardware, or/and on software levels.

The known methods (see, e.g., [6 - 13]) for providing fault tolerance use hardware or time redundancy, which increases the cost or degrades the performance of computational systems. Therefore, they are few suitable for real-time computing systems and parallel processors. The algorithm-based fault tolerance (ABFT) methods [14 - 19] are much more suitable for such systems. 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 very easily. The modified algorithm will take more time to operate on the encoded data when compared to the original algorithm. This time overhead must not be excessive. Thus, ABFT methods establish the modification rules of the original (applied) algorithms and input data arrays.

An known ABFT method called weighted checksum (WCS) one, which is specially tailored for matrix algorithms and array architectures, has been proposed by Abraham et al. [15, 16]. However, the original WCS method is few suitable for such important algorithms as Gaussian elimination, Choleski, Jordan-Gauss and Faddeev algorithms, etc., since a single transient fault in processor or processor elements (PE) of an array during computation might cause multiple output errors, which can not be located. Therefore in the papers [18, 19], we improve Abraham's WCS method for the LU-decomposition and extend it for the $LL^T$-decomposition, linear systems solution and matrix inversion, i.e. we propose ABFT versions of above mention algorithms, which are based on Gauss elimination. In particular, the proposed version of fault-tolerant Gauss elimination algorithm enables to correct up to $N^2/2$ single errors during time of solving of the whole triangularisation problem.

By means above mention algorithms can be effectively solved many main linear algebra tasks. However, as much or many linear algebra tasks such as least square problem, singular value decomposition, eigen values and eigenvectors problems, matrix pseudoinverse, etc., can't be solved by means these algorithms. In this case must be applied more complicated algorithms based on matrix QR-decomposition.

Householder’s method of reflections [1, 2] is a most simple among known methods of QR-decomposition: its computational complexity is almost equal $2N^3/3$ multiplication with additional
Fault Tolerant QR - decomposition Algorithm Based on Householder’s method

operations (for real matrix $A(N,N)$). Therefore in this paper, we design fault-tolerant version QR -
decomposition algorithm based on Householder’s method of reflections and modified weighted
checksum method. The derived algorithm enables to correct a single error among elements of any
column (or row) of an input matrix $A(M,N)$ occurred on any from $N$ steps of algorithm
implementation. Consequently, it is possible to correct up to $N$ single errors during solving the whole
decomposition task. This effect is obtained by increasing the computational complexity of the original
Householder’s method on $8.5N^2 + O(N)$ multiply-add operations. Finally, the parallel version of
proposed algorithm destined for realisation on a fixed-size linear processor array with fully local
communications and low I/O requirements is designed.

2 Fault model and weighted checksum code.

Module-level faults are assumed [15] in the algorithm-based fault tolerance. A module (processor or
PE for parallel computers here) 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 WCS code has been adopted by Jou and Abraham [15] 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 named a check element. Information is compressed in
such a way that it is preserved during algorithm implementation. 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 correction process is much
smaller than that of the original computation. For example, a WCS encoded data vector $a(N)$ with
Hamming distance equal three which can correct a single error (SEC) can be expressed as

$$A^T = [a_1 \ a_2 \ ... \ a_N \ \text{PCS} \ \text{QCS}], \quad (1)$$

where $a_i$ is a element of a data vector $a(N)$,

$$\text{PCS} = p^T \cdot a(N) \ \text{and} \ \text{QCS} = q^T \cdot a(N), \quad (2)$$

and $p(N)$, $q(N)$ - are encoder vectors.

Possible choices for vector pairs $p$ and $q$ are, for example, [15]

$$p^T = [1 \ 1 \ ... \ 1] \ \text{and} \ q^T = [2^0 \ 2^1 \ ... \ 2^{N-1}] \quad (3)$$

(were $q$ is named exponential weighted encoder vector), or [20]

$$p^T = [1 \ 1 \ ... \ 1] \ \text{and} \ q^T = [1 \ 2 \ 3 \ ... \ N] \quad (4)$$

(were $q$ is named linear weighted encoder vector).

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

Based on the encoding vector, a matrix $A(M,N)$ can be encoded as either

a row encoded matrix $A_R$ given by

$$A_R = [A \ \ A^*p(N) \ \ A^*q(N)] = [A \ \ \text{PCS} \ \ \text{QCS}], \quad (5)$$

a column encoded matrix $A_C$, where

$$\begin{bmatrix}
A \\
A
\end{bmatrix}$$

Parallel Numerics’97/ Zakopane, Poland / September 5-7, 1997
Fault Tolerant QR - decomposition Algorithm Based on Householder’s method

\[ A_C = \begin{bmatrix} p(M)*A \\ q(M)*A \end{bmatrix} = \begin{bmatrix} PCS \\ QCS \end{bmatrix}, \quad (6) \]

or a full encoded matrix \( A_{RC} \) [20, 21] given by

\[ A_{RC} = \begin{bmatrix} A_R \\ p(M)*A \\ q(M)*A \end{bmatrix} = \begin{bmatrix} A_R \\ PCS \\ QCS \end{bmatrix}. \quad (7) \]

For example, for matrix multiplication \( A(M,N) * B(N,K) = C(M,K) \), the column encoded matrix \( A_C \) of form (6) is exploited [20]. Then choosing the linear weighted vectors pare (4), the following equation is computed:

\[ A_C * B = C_C. \quad (8) \]

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

\[ S_1 = \sum_{i=1}^{N} c_{ij} \cdot PCS_j \quad (9) \]

and

\[ S_2 = \sum_{i=1}^{N} i \cdot c_{ij} \cdot QCS_j \quad (10) \]

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

- if \( S_1 = S_2 = 0 \) then no error has been detected;
- if \( S_1 \neq 0 \) and \( S_2 = 0 \) then \( PCS_j \) is inconsistent;
- if \( S_2 \neq 0 \) and \( S_1 = 0 \) then \( QCS_j \) is inconsistent;
- if \( S_1 \neq 0 \) and \( S_2 \neq 0 \) then \( S_2 / S_1 = i \) and element \( c_{ij} \) is erroneous,

and the correction procedure is:

\[ c_{ij} = c_{ij} - S_1. \quad (11) \]

3 Numerical properties of WCS method in the case of floating point realisation.

In [22] the numerical properties of SEC codes based on linearly and exponentially weighted encoder vectors was considered in detail for case of ABFT floating - point implementation. The main result is next. In the cases when encoder vectors (3) and (4) are applied respectively \( \log_2 N \) and \( \log_2(\log_2 N) \) extra bits are needed to provide numerical accuracy of computation, i.e. to provide that no false alarms occur in worst-case round errors. Therefore both (3) and (4) encoder vectors are few suitable for ABFT floating-point realisation.

In [16] the three various stages of ABFT technique which are prone to numerical errors were identified: the coding phase, actual data computation phase, and error correction phase. Note, that the round errors of second stage is determined only numerical properties of applied algorithm (Householder’s algorithm here). For example, in [21] the value of tolerance \( \tau \) (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 \( \tau \) ) was determined for case of floating point implementations of LU- and QR-decomposition algorithms. In particular, it was be shown, that the value of tolerance \( \tau \) is necessarily large for the LU - decomposition and Gaussian elimination with pairwise pivoting, but is acceptably small for the QR-decomposition.

It has been proved [23] that the maximum round error during of data vector \( a(N) \) coding, is given by

\[ E \leq e^{\cdot \delta} \cdot \left| a \right|^2 \cdot \left| x \right|^2, \quad (12) \]
where \( x(N) \) - encoder vector, \( \| \cdot \|_2 \) - Euclidean norm of vector, \( \varepsilon \) and \( \delta \) - machine dependent parameters. Thus, the straightforward way to reduce the amount of error during coding is to minimise \( \| x \|_2 \). However, it has been shown in [16], that we cannot select an \( \| x \|_2 \) as small as we wish because such vector \( x \) would not have the high reflectively, i.e. that the ratio of the change in the code value to the change in the data element would very small. High reflectivity of a code is essential in the error correction phase, because if the reflectivity of code is very small, two errors in the data element, which are almost equal in value, but at the same time distinctly observable, will reflect the same amount of error in the check element which makes the discrimination of these two errors very difficult. Therefore, the strategy should be to make a compromise in selecting the norm of encoder vector which will give a small, and at the same time, will have moderately high reflectivity. What is why in [16] were presented some examples of experimental designed encoder vectors which have small value of Euclidean norm and high value of reflectivity. Its are, in particular, next:

1) average and weighted average encoder vectors:

\[
p^T = \begin{bmatrix} 1/N & 1/N & \ldots & 1/N \end{bmatrix} \quad \text{and} \quad q^T = \begin{bmatrix} 1/N & 2/N & \ldots & N/N \end{bmatrix}
\]

2) normalised encoder vectors

a) for vector \( a(N) \)

\[
p^T = \begin{bmatrix} c/\| a \|_2 & c/\| a \|_2 & \ldots & c/\| a \|_2 \end{bmatrix}
\]

b) for matrix \( A(M,N) \)

\[
q^T = \begin{bmatrix} c/\| A \| & c/\| A \| & \ldots & c/\| A \| \end{bmatrix},
\]

where \( \| A \| \) is the average value of matrix columns (or rows) Euclidean norms and \( c \) is a constant fixed by user.

Experimental evaluation of numerical error for proposed encoder vectors also were conducted in [16]. The main result are next: when round errors are the larger problem, one should use normalised encoder vectors; for overflow problems, one should use average encoder vectors.

### 4 Design of the ABFT Householder’s algorithm.

Householder’s method of reflections [1, 2] is a most simple among known methods of matrix QR-decomposition: its computational complexity is almost equal \( 2N^3/3 \) multiplication with additional operations (for real matrix \( A(N,N) \)). Besides, it is a base for many other linear algebra algorithms such as orthogonal degree algorithm for solution of eigen values and eigenvectors problems, modified Faddeeva [24] and orthogonal Jordan-Gauss [25] algorithms for matrix inverse and pseudoinverse, solution of linear systems and least square problem, etc.

Besides, Householder’s method is the method of equivalent matrix transformation, i.e. the input matrix norm is constant during its implementation. For case of fault tolerant realisation it is important, because we can apply the pare of normalised encoder vectors (15) for the coding of input matrix \( A \), and during algorithm implementation don’t to compute a new average value of matrix columns (or rows) norms \( \| A \| \). Note, that in this case the maximum round error \( E \) (see (12)) is constant during algorithm implementation. Therefore in this paper we design fault-tolerant version QR-decomposition algorithm based on Householder’s method of reflections and modified weighted checksum method with normalised encoder vectors.

The algorithm given by construction (16) corresponds to the original Householder’s algorithm of QR-decomposition. Note that the input matrix \( A(M,N) = A^1 = \{ a_{ij} \} \) is recursively modified during \( K = (M-1) \) or \( K = N \) (if \( M \leq N \) or \( M > N \) respectively) of computation steps to obtain the upper triangular matrix \( R = A^{K+1} \). Thus, each \( i \)-th step of algorithm includes two phases. The first phase is consist from the computation of the \( i \)-th column norm \( \sigma' \), the calculation of the values \( \alpha^i, \beta^i \) and the new value of the element \( a_{ii} = v_{ii} \), and the computation of inner product \( y_k^i \) of \( i \)-th and \( k \)-th columns of
Fault Tolerant QR - decomposition Algorithm Based on Householder’s method __________ PACT __

matrix A (with new value of the element $a_{ii}^i$). The second phase includes the computation of the new values of elements $a_{jk}^{i+1}$ and resulting elements $r_{ii}$ of i-th row of matrix R. Note also, that instruction $y_{i}^i := \sigma^i - \alpha^i \cdot a_{ii}^i$ was included in construction (16) only for using (in following) at the error detection and correction procedures.

for $i:=1$ to $K$ do
    begin
    { Phase 1: the computation of i-th column norm }
        $\sigma^i := 0$;
        for $j := i$ to $M$ do
            $\sigma^i := \sigma^i + a_{ji}^i \cdot a_{ji}^i$;
        if $a_{ii}^i < 0$ then $\alpha^i := \text{SQRT}(\sigma^i)$
        else $\alpha^i := -\text{SQRT}(\sigma^i)$;
        $v_{ii}^i := a_{ii}^i - \alpha^i$;
    { Phase 1: the computation of the inner product of i-th and k-th columns of matrix $A^i$ }
        $y_{ik}^i := \sigma^i - \alpha^i \cdot a_{ii}^i$;
        $a_{ii}^i := v_{ii}^i$;
        for $k := i+1$ to $N$ do
            begin
                $y_{ik}^i := 0$;
                for $j := i$ to $M$ do
                    $y_{ik}^i := y_{ik}^i + a_{ji}^i \cdot a_{jk}^i$;
                $y_{ik}^i := 2 \cdot \beta^i \cdot y_{ik}^i$;
            end;
    { Phase 2: the computation of the elements of matrix $A^{i+1}$ }
        for $k := i+1$ to $N$ do
            begin
                for $j := i$ to $M$ do
                    $a_{jk}^{i+1} := a_{jk}^i + a_{ji}^i \cdot y_{ik}^i$;
            end;
    { Phase 2: the computation of the elements of i-th row of the resulting matrix R }
        $r_{ii} := \alpha^i$;
    end;
end;

From construction (16) it is shown, that if during the i-th step of computation ($i=1,\ldots,K$) the element $y_{ik}^i$ is wrongly calculated, then errors will appear in all the elements $a_{ik}^{i+1}$ of the k-th column ($k=i+1,\ldots,N$) of matrix $A^{i+1}$. Moreover, if any element $\beta^i$, $\sigma^i$, $\alpha^i$ or $a_{ii}^i$ is wrongly calculated, then errors appear in firstly in all the elements $y_{ik}^i$ ($k=i+1,\ldots,N$) and then in the all elements of matrix $A^{i+1}$. In both cases, these errors can not be corrected by the original WCS method. Besides, if at the (i-1)-th step of algorithm implementation any element $a_{jk}^i$ was wrongly calculated, then at the i-th step error firstly will appear in the element $y_{ik}^i$ and it will cause of above mention multiple output errors, which also can not be located. If the check and correction the all calculated elements are performed during computation, then the complexity of the original algorithm increases more than twice. Thus, the original WSC method isn’t suitable for algorithm (16).

For removing of the mentioned above defects by means modification of the original WCS method, the following confirmations and theorems are proved. Note, that it is assumed, that one single transient error may be occurred at the each phase of any step of algorithm (16). However, if the error is occurred at the phase 2 of i-th step then next error shouldn’t be occurred at the phase 1 of (i+1)-th step of algorithm implementation.
Confirmation 1. If during the (i-1)-th step of computations the element $a_{ik}$ was wrongly calculated, then errors will not appear anywhere among the other elements of matrix $A^i$.

The proof of this confirmation follows directly from construction (16), in which any element $a_{ik}$ don’t takes part in calculating of remaining elements of matrix $A^i$.

Consequence 1. For any $i$-th step of algorithm it is possible don’t check and correct the new values of elements $a_{ik}$ during phase 2 implementation.

Confirmation 2. If during the (i-1)-th step of computations the element $a_{ik}$ was wrongly calculated, then at the phase 1 of the i-th step the error will appear only in the element $y_{ik}$ (if $i < k$) or consistently in the elements $\alpha^i$, $\alpha^i$, $\beta^i$, $v_{ii}$ and $y^i_k$ (if $i = k$).

The proof of this confirmation follows directly from construction (16).

Consequence 2. For any step of algorithm it is possible before executing of the phase 1 to check and to correct only the elements of the i-th (pivoting) column of the matrix $A^i$ and after executing of the phase 1 to check and to correct only the erroneous values $y^i_k$ ($k=i+1,...,N$). Besides, before calculation of the elements $y^i_k$ we must to check and to correct the elements $\beta^i$, $v_{ik}$ and $y^i_k$.

Note, that if value $y^i_k$ is erroneous then either the error was appeared during computation of $y^i_k$ at the phase of i-th step of algorithm, or the error was appeared during computations the element $a_{ik}$ at the phase 2 of the (i-1)-th step of algorithm. Therefore, in the both cases we should be to check (and may be to correct) the elements of $k$-th column of matrix $A^i$ (if value $y^i_k$ is erroneous) before executing of the phase 2 at the i-th step of algorithm.

Now, the procedures of errors detection and correction of above mention values should be considered. The following theorems are proved for this.

Theorem 1. Let an element $y^i_k$ ($k=i+1,...,N$) was wrongly calculated during executing of the phase 1 at the i-th step of algorithm (16). Then it is possible to correct its value using the WCS method for the row encoded matrix $A_R = [A \quad A*p(N) \quad A*q(N)] = [A \quad PCS(M) \quad QCS(M)]$ (5) after executing of the phase 1.

Proof. Without the loss of generality, we assumed that the following encoder vector pare are used:

$$p^T = [1/|A| \quad 1/|A| \quad ... \quad 1/|A|]$$

and

$$q^T = [1/|A| \quad 2/|A| \quad ... \quad N/|A|].$$

Then according to (5) and (16), at the beginning of the i-th step of algorithm, the values of elements $PCS^i_j$ and $QCS^i_j$ ($j=i,...,M$) of vectors $PCS(M)$ and $QCS(M)$ are equal:

$$PCS^i_j = a_{ij}/|A| + a_{i+1j}/|A| + ... + a_{nj}/|A|$$

and

$$QCS^i_j = 1-a_{ij}/|A| + 2a_{i+1j}/|A| + ... + N\cdot a_{nj}/|A|.$$ 

After performing the first phase the values $y^i_j$ and $y^i_k$ become as follows:

$$y^i_j := \sigma^i - \alpha^i \cdot a_{ii} = (a_{ii} - a_{i+1i} + a_{i+1i} \cdot a_{ii+1} + ... + a_{Mi} \cdot a_{Mi}) - \alpha^i \cdot a_{ii},$$

consequently,

$$y^i_k := (a_{ik} - \alpha^i) \cdot a_{ik} + a_{i+1i} \cdot a_{i+1k} + ... + a_{Mi} \cdot a_{Mi} = v_{ii} \cdot a_{ii} + a_{i+1i} \cdot a_{i+1k} + ... + a_{Mi} \cdot a_{Mi};$$

and for all $k=i,...,N$ the values $y^i_k$ are equal:

$$y^i_k = v_{ii} \cdot a_{ik} + a_{i+1i} \cdot a_{i+1k} + ... + a_{Mi} \cdot a_{Mi}.$$ 

Besides this, after performing the first phase of i-th step of algorithm with matrix $A_R$ the values $y^i_{PCS}$ and $y^i_{QCS}$ become as follows:

$$y^i_{PCS} = v_{ii} \cdot PCS^i_l + a_{i+1i} \cdot PCS^i_{l+1} + ... + a_{Mi} \cdot PCS^i_{M},$$

$$y^i_{QCS} = v_{ii} \cdot QCS^i_l + a_{i+1i} \cdot QCS^i_{l+1} + ... + a_{Mi} \cdot QCS^i_{M}.$$
Fault Tolerant QR - decomposition Algorithm Based on Householder’s method

\[ y_{QCS}^i = v_i \cdot \text{QCS}_{i}^i + a_{i+1}^i \cdot \text{QCS}_{i+1}^i + \ldots + a_{M_i}^i \cdot \text{QCS}_{M_i}^i. \]  (24)

On the other side, in according to the error detection and correction procedure (see (9), (10)) the values \( y_{PCS}^i \) and \( y_{QCS}^i \) should be calculated in following way:

\[ y_{PCS}^i = y_i^i / |A| + y_{i+1}^i / |A| + \ldots + y_{N}^i / |A|, \]  (25)

\[ y_{QCS}^i = i \cdot y_i^i / |A| + (i+1) \cdot y_{i+1}^i / |A| + \ldots + N \cdot y_N^i / |A|, \]  (26)

and then the values (23) and (24) are should be compared with the values (25) and (26) respectively. If these pares of values are equal then no error has been detected.

It is below shown that the value \( y_{PCS}^i \) calculated by expression (23) should be equal the value \( y_{PCS}^i \) calculated by expression (25).

\[ y_{PCS}^i = v_i \cdot \text{PCS}_{i}^i + a_{i+1}^i \cdot \text{PCS}_{i+1}^i + \ldots + a_{M_i}^i \cdot \text{PCS}_{M_i}^i = v_i \cdot (a_i^i / |A| + a_{i+1}^i / |A| + \ldots + a_{N}^i / |A|) + \\
+ a_{i+1}^i \cdot (a_i^i / |A| + a_{i+1}^i / |A| + \ldots + a_{N}^i / |A|) + \ldots + a_{M_i}^i \cdot (a_i^i / |A| + a_{i+1}^i / |A| + \ldots + a_{N}^i / |A|) + \\
+ a_{i+1}^i \cdot (a_{i+1}^i / |A| + \ldots + a_{N}^i / |A| / |A|) = (v_i \cdot a_i^i + a_{i+1}^i \cdot a_{i+1}^i + \ldots + a_{M_i}^i \cdot a_{M_i}^i) + (v_i \cdot a_{i+1}^i + a_{i+2}^i \cdot a_{i+1}^i + \ldots + a_{M_i}^i \cdot a_{M_i}^i) + \\
+ (v_i \cdot a_{i+1}^i + a_{i+1}^i \cdot a_{i+1}^i + \ldots + a_{M_i}^i \cdot a_{M_i}^i) / |A| = y_i^i / |A| + y_{i+1}^i / |A| + \ldots + y_{N}^i / |A|. \]

The proof of equality of the values \( y_{QCS}^i \) calculated by expression (24) and (26) are performed in similar way. The theorem 1 is proved.

Thus, the procedure (27) of error detection and correction in values \( y_{QCS}^i \) is consisted from calculation of expressions (25) and (26) after performing each phase 1 of algorithm (16) and then performing of the following steps:

- if \( S_1 = 0 \) and \( S_2 = 0 \) then no error has been detected;
- if \( S_1 \neq 0 \) and \( S_2 = 0 \) then \( y_{PCS}^i \) is inconsistent; \hspace{1cm} (27)
- if \( S_2 \neq 0 \) and \( S_1 = 0 \) then \( y_{QCS}^i \) is inconsistent;
- if \( S_1 \neq 0 \) and \( S_2 \neq 0 \) then \( S_2 / S_1 = k \) and element \( y_{k}^i \) is erroneous,

and the correction procedure is:

\[ y_{k}^i = y_{k}^i - S_1 \cdot |A|, \]

were \( S_1 \) and \( S_2 \) are subtractions of the results of expression (25), (23) and (26), (24) respectively.

It was above noted, that if value \( y_{k}^i \) is erroneous then either the error was appeared during computation of \( y_{k}^i \) at the phase 1 of i-th step of algorithm, or the error appeared during computations the element \( a_{k}^i \) at the phase 2 of the (i-1)-th step of algorithm. Therefore, in the both cases we should be to check (and may be to correct) the elements of k-th column of matrix A (in which value \( y_{k}^i \) is erroneous) before executing of the phase 2 at the i-th step of algorithm (k=i+1,..,N).

Now we should derive a procedure for the detection and correction of erroneous elements in k-th column of matrix A.

**Theorem 2.** Let an element \( a_{k}^{i+1} \) was wrongly calculated during executing of the phase 2 at the i -th step of algorithm (16). Then it is possible to correct its value using the WCS method for the column encoded matrix (6)

\[
A_C = \begin{bmatrix}
A \\
p(M)A \\
q(M)A \\
\end{bmatrix} = \begin{bmatrix}
A \\
PRS(N) \\
QRS(N) \\
\end{bmatrix} \quad (28)
\]

after executing of this phase.
Proof. In according to (6) and (16), at the beginning of the i-th step of algorithm, the values of elements $PRS_i^k$ and $QRS_i^k \ (k=i, \ldots, N)$ of vectors $PRS(N)$ and $QRS(N)$ are equal:

$$PRS_i^k = a_{ik}^i / |A| + a_{i+1k}^i / |A| + \ldots + a_{mk}^i / |A|$$

and

$$QRS_i^k = i.a_{ik}^i / |A| + (i+1).a_{i+1k}^i / |A| + \ldots + M.a_{mk}^i / |A|,$$

and its values don’t compute at the first phase of i-th step for $k=i+1, \ldots, N$.

Let after performing the first phase the values $PRS_i^1$ and $QRS_i^1$ are become as follows:

$$PRS_i^1 = PRS_i^1 - \alpha^i.a_{ii}^i / |A| \quad \text{and} \quad QRS_i^1 = QRS_i^1 - i.\alpha^i.a_{ii}^i / |A|.$$  

Consequently,

$$PRS_i^1 = v_i / |A| + a_{i+1i}^i / |A| + \ldots + a_{mi}^i / |A|$$

and

$$QRS_i^1 = i.v_i / |A| + (i+1).a_{i+1i}^i / |A| + \ldots + M.a_{mi}^i / |A|.$$  

Then in according to (16) after performing the second phase of i-th step of algorithm with matrix $A_C$, the values $PRS_k^i$ and $QRS_k^i \ (k=i+1, \ldots, N)$ become as follows:

$$PRS_k^{i+1} := PRS_k^i + PRS_i^1 \cdot y_k^i \quad (34)$$

and

$$QRS_k^{i+1} := QRS_k^i + QRS_i^1 \cdot y_k^i \quad (35)$$

On the other side, in according to the error detection and correction procedure (see (9), (10)) these values should be calculated in following way:

$$PRS_k^{i+1} = a_{ik}^{i+1} / |A| + a_{i+1k}^{i+1} / |A| + \ldots + a_{mk}^{i+1} / |A|,$$

and

$$QRS_k^{i+1} = i.a_{ik}^{i+1} / |A| + (i+1).a_{i+1k}^{i+1} / |A| + \ldots + N.a_{mk}^{i+1} / |A|,$$

were $a_{ik}^{i+1} := a_{ik}^i + a_{i+1i}^i \cdot y_k^i$ (see construction (16)).

Then in according to the error detection and correction procedure the values (34) and (35) are should be compared with the values (36) and (37) respectively. If these pares of values are equal then no error has been detected.

It is below shown that the value $PRS_k^{i+1}$ calculated by expression (34) should be equal the value $PRS_k^{i+1}$ calculated by expression (36).

$$PRS_k^{i+1} = a_{ik}^{i+1} / |A| + a_{i+1k}^{i+1} / |A| + \ldots + a_{mk}^{i+1} / |A| = (a_{ik}^i + v_i \cdot y_k^i) / |A| + (a_{i+1k}^i + a_{i+1i}^i \cdot y_k^i) / |A| + \ldots + (a_{mk}^i + a_{mi}^i \cdot y_k^i) / |A| = PRS_k^i + PRS_i^1 \cdot y_k^i.$$  

The proof of equality of the values $QRS_k^{i+1}$ calculated by expression (35) and (37) are performed in similar way. The theorem 2 is proved.

Thus, the procedure of error detection and correction of element $a_{ik}^i$ is consisted from:

1) calculation at the first phase of each i-th step the values $PRS_i^1$ and $QRS_i^1$ in according to expression (31);

2) after detection and correction of erroneous value $y_k^i$ in according to procedure (27) - calculation of expressions (36) and (37) only for k-th column of matrix $A_i$ after performing each phase 1 of algorithm (16);
3) performing of the following procedure:
   if $S_1 = 0$ and $S_2 = 0$ then no error has been detected;
   if $S_1 \neq 0$ and $S_2 = 0$ then $\text{PRS}_k^i$ is inconsistent;
   if $S_2 \neq 0$ and $S_1 = 0$ then $\text{QRS}_k^i$ is inconsistent;
   if $S_1 \neq 0$ and $S_2 \neq 0$ then $S_2 / S_1 = j$ and element $a_{jk}^i$ is erroneous,

   and the correction procedure is:
   $$a_{jk}^i = a_{jk}^i - S_1 \cdot |A|,$$

   were $S_1$ and $S_2$ are subtractions of the results of expression (36), (34) and (37), (35) respectively.

   In accord to confirmation 2 and theorem 2, before executing of i-th step of algorithm we should be certain that the elements of the i-th (pivoting) column of the matrix $A^i$ were calculated correctly at the (i-1) step of algorithm. Besides, before calculating of the elements $y_{ik}^i$ (k=i+1,...,N) we should be certain that the elements $\sigma^i$, $\alpha^i$, $\beta^i$, $v_{ii}$ and $y_{1i}^i$, $\text{PRS}_{i}^i$, $\text{QRS}_{i}^i$ also aren’t erroneous. For deriving this we proposed to use any method of time redundancy, for example, triple time redundancy (TTR) method [7], because this time must not be large.

   As a result, the ABFT Householder’s algorithm will consist in executing the following stages:

   1) The original matrix $A(M,N) = \{a_{jk}\}$ is represented in the form of full encoded matrix 
   (7) $A_{RC}^{M+2,N+2} = \{a_{jk}^{1}\}$, in which
   $$a_{jk}^{1} = a_{jk} \quad (\text{for } j=1,...,M; k=1,...,N) ,$$
   $$a_{M+1k}^{1} = \text{PRS}_{k}^i , a_{M+2k}^{1} = \text{QRS}_{k}^i \quad (\text{for } k=1,...,N+2)$$
   and
   $$a_{jN+1}^{1} = \text{PCS}_{j}^i , a_{jN+2}^{1} = \text{QCS}_{j}^i \quad (\text{for } j=1,...,M) .$$

   2) For $i = 1,2, \ldots, K$, stages 3-9 are repeated.

   3) At the beginning of the i-th step of the algorithm, the error detection and correction within
   elements belonging to the i-th column of $A^i$ are performed. This stage needs to execute
   approximately 2(M-i) multiply-add operations.

   4) The values of elements $\sigma^i$, $\alpha^i$, $\beta^i$, $v_{ii}$ and $y_{1i}^i$, $\text{PRS}_{i}^i$, $\text{QRS}_{i}^i$ are three times calculated. This stage
   needs to execute approximately 3(M-i) multiply-add operations and 6 operations of square rooting.

   5) New values of elements $y_{ik}^i$ column are computed (k=i+1,...,N+2).

   6) The error detection and correction within the computed elements $y_{ik}^i$ are performed. This stage
   requires approximately 2(N-i) multiply-add operations.

   7) The error detection and correction within the elements $a_{jk}^i$ of the erroneous k-th column are
   performed. This stage requires approximately 2(N-i) multiply-add operations.

   8) The elements $a_{jk}^{i+1}$ of the matrix $A_{i+1}^i$ are calculated.

   9) The error detection and correction of the elements of i-th row of the resulting matrix $R$ are
   performed. This stage requires approximately 2(N-i) multiply-add operations.

   In Pascal-like form the ABFT Householder’s algorithm may be written as following
   construction, were $A_1 = 1/|A|$, $A_J = j/|A|$ and $A_K = k/|A| :
   for $i=1$ to $K$ do
   begin
   \{ Phase 1: errors detection and correction within elements of the i-th column of $A^i \}
   \begin{align*}
   \text{PRS}_{i}^i & := 0; \quad \text{QRS}_{i}^i := 0; \\
   \text{for } j & := i \text{ to } M \text{ do begin } \text{PRS}_{i}^i := \text{PRS}_{i}^i + a_{ji}^i \cdot A_1; \quad \text{QRS}_{i}^i := \text{QRS}_{i}^i + a_{ji}^i \cdot A_J; \text{ end; } \quad \text{S}_1 := \text{PRS}_{i}^i - a_{M+1j}^i; \quad \text{S}_2 := \text{QRS}_{i}^i - a_{M+2j}^i; \\
   \text{if } S_1 < 0 \text{ and } S_2 = 0 \text{ then } a_{M+1j}^i := \text{PRS}_{i}^i; \quad \text{if } S_2 < 0 \text{ and } S_1 = 0 \text{ then } a_{M+2j}^i := \text{QRS}_{i}^i; \\
   \text{if } S_1 < 0 \text{ and } S_2 > 0 \text{ then begin } j := S_2/S_1; \quad a_{ji}^i := a_{ji}^i - S_1 \cdot |A|; \text{ end; } \quad \text{end; }
   \end{align*}
   \{ Phase 1: the three times computation of i-th column norm \}

Parallel Numerics 97/ Zakopane,Poland / September 5-7, 1997
for s := 1 to 3 do
begin
\(\sigma_i(s) := 0;\)
for j := i to M do \(\sigma_i(s) := \sigma_i(s) + a_{ji} \cdot a_{ii};\)
end
if \(\sigma_i(1) = \sigma_i(2)\) or \(\sigma_i(1) = \sigma_i(3)\) then \(\sigma_i := \sigma_i(1)\)
else \(\sigma_i := \sigma_i(2);\)

{ Phase 1: the three times computation of the elements \(\alpha_i, \beta_i, v_{ii}, y_{ii} \) and \(a_{M+1i}, a_{M+2i}\) }
for s := 1 to 3 do if \(a_{ii} < 0\) then \(\alpha_i(s) := \sqrt{\sigma_i}\)
else \(\alpha_i(s) := -\sqrt{\sigma_i};\)
if \(\alpha_i(1) = \alpha_i(2)\) or \(\alpha_i(1) = \alpha_i(3)\) then \(\alpha_i := \alpha_i(1)\)
else \(\alpha_i := \alpha_i(2);\)

for s := 1 to 3 do \(\beta_i(s) := -1/\sqrt{2 \cdot \sigma_i + 2 \cdot \alpha_i \cdot a_{ii}};\)
if \(\beta_i(1) = \beta_i(2)\) or \(\beta_i(1) = \beta_i(3)\) then \(\beta_i := \beta_i(1)\)
else \(\beta_i := \beta_i(2);\)

for s := 1 to 3 do \(v_{ii}(s) := a_{ii} - \alpha_i;\)
if \(v_{ii}(1) = v_{ii}(2)\) or \(v_{ii}(1) = v_{ii}(3)\) then \(v_{ii} := v_{ii}(1)\)
else \(v_{ii} := v_{ii}(2);\)

for s := 1 to 3 do \(y_i(s) := \sigma_i - \alpha_i \cdot a_{ii};\)
if \(y_i(1) = y_i(2)\) or \(y_i(1) = y_i(3)\) then \(y_i := y_i(1)\)
else \(y_i := y_i(2);\)

for j := M+1 to M+2 do
for s := 1 to 3 do
begin
\(a_{ji}(s) := a_{ji}(s) - \alpha_i \cdot a_{ii} \cdot AI;\)
if \(a_{ji}(1) = a_{ji}(2)\) or \(a_{ji}(1) = a_{ji}(3)\) then \(a_{ji} := a_{ji}(1);\)
else \(a_{ji} := a_{ji}(2);\)
end
\(a_{ii} := v_{ii};\)

{ Phase 1: the computation of the inner product of i-th and k-th columns of matrix A^i } 
for k := i + 1 to N + 2 do
begin
\(y_{ik} := 0;\)
for j := i to M do
\(y_{ik} := y_{ik} + a_{ji} \cdot a_{ki};\)
\(y_{ik} := 2 \cdot \beta_i \cdot y_{ik};\)
end

{ Phase 1: the error detection and correction within the computed elements y_{ik} }
\(y_{PCS} := 0;\) 
\(y_{QCS} := 0;\)
for k := i+1 to N do begin \(y_{PCS} := y_{PCS} + y_{ik} \cdot AI;\) \(y_{QCS} := y_{QCS} + y_{ik} \cdot AK;\) end;
\(S_1 := y_{PCS} - y_{N+1};\) \(S_2 := y_{QCS} - y_{N+2};\)
if \(S_1 < 0\) and \(S_2 < 0\) then \(y_{N+1} := y_{PCS};\)
if \(S_2 < 0\) and \(S_1 < 0\) then \(y_{N+2} := y_{QCS};\)
if \(S_1 < 0\) and \(S_2 < 0\) then begin \(k := S_2/S_1;\) \(y_{ik} := y_{ik} - S_1 \cdot |A|;\) end;
{ Phase 2: the computation of the elements of matrix A^{i+1} }
for k := i + 1 to N + 2 do
for j := i to M + 2 do
\[
a_{ik}^{i+1} = a_{ik}^i + a_{ji}^i \cdot y_k^i;
\]

{Phase 2: the computation and correction of the elements of i-th row of the resulting matrix \( R \)}

\[
\text{PCS}_{ik}^i := 0; \quad \text{QCS}_{ik}^i := 0;
\]

For \( k := i \) to \( N \) do begin

\[
r_{ik} := a_{ik}^{i+1};
\]

\[
\text{PCS}_{ik}^i := \text{PCS}_{ik}^i + a_{ik}^{i+1} \cdot A_l; \quad \text{QCS}_{ik}^i := \text{QCS}_{ik}^i + a_{ik}^{i+1} \cdot A_k;
\]

end;

\[
S_1 := \text{PCS}_{ik}^i - a_{N+1i}^{i+1}; \quad S_2 := \text{QCS}_{ik}^i - a_{N+2i}^{i+1};
\]

If \( S_1 < > 0 \) and \( S_2 = 0 \) then \( a_{N+1i}^{i+1} := \text{PCS}_{ik}^i \);

If \( S_2 < > 0 \) and \( S_1 = 0 \) then \( a_{M+2i}^{i+1} := \text{QCS}_{ik}^i \);

If \( S_1 < > 0 \) and \( S_2 < > 0 \) then begin \( k := S_2 / S_1 \); \( r_{ik} := r_{ik} - S_1 \cdot |A| \); end;

\[
r_{ii} := \alpha_i;
\]

end;

From comparison of the constructions (16) and (38) it is shown, that due to inserting the procedures of error detection and correction the computational complexity of algorithm (38) increases at the each i-th step approximately on the \( 5(M-i) + 6(N-i) \) operations of multiplication with addition. This means, that the computational complexity of the whole algorithm increases (for case \( M = N \)) on \( 5.5 \cdot N^2 + O(N) \) multiply-add operations. Besides, due to increasing of input matrix sizes, the computational complexity of the proposed algorithm (38) is yet increased (for case \( M = N \)) on \( 3 \cdot N^2 + O(N) \) multiply-add operations. As a result, the computational complexity of the whole ABFT version of Householder’s algorithm is increased approximately on \( 8.5 \cdot N^2 + O(N) \) multiply-add operations in comparison with original algorithm (16).

However, the proposed AFT version of the Householder’s algorithm enables to correct one single transient error among elements of any 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 \( K \) single errors during solving the whole decomposition task.

5 Design of the parallel version of the ABFT Householder’s algorithm and architecture the linear fixed size processor array for its realisation.

A basic requirement in practical system designs for linear algebraic problems is an ability to process large size matrices on processor arrays with a fixed number of PEs [3, 27 - 29]. To provide this ability, two partitioning methods [3] are usually used: locally sequential globally parallel (LSGP) method and locally parallel globally sequential (LPGS) method. Both of them are based on the decomposition of a dependence graph (DG) of an algorithm into a set of regular subgraphs, but differ in the way how these subgraphs are mapped onto resulting structural schemes. In the LSPG method, one subgraph is mapped to one PE, and each PE sequentially executes the nodes of the corresponding subgraph. Therefore, an additional local memory within each PE is needed. To avoid this disadvantage, one subgraph is mapped to one array in the LPGS method. All nodes within one subgraph are processed concurrently, while all subgraphs are processed sequentially. As a result, all intermediate data which correspond to data dependencies between subgraphs should be stored in buffers outside the processor array. Based on our approach to mapping recursive algorithms into processor arrays [27], we employ this scheme in order to implement the ABFT versions of Householder’s algorithm on a linear array with \( n < N \) PEs, where \( n \) is a fixed number. Firstly, we consider the DG for the proposed algorithm.

The two-dimensional DG \( G1 \) of the ABFT Householder’s algorithm (38) is shown in Fig.1 for case \( M > N \) and \( N = 4 \), were \( a(i) \) and \( r(j) \) are the i-th column \((i=1,\ldots,N)\) of input matrix \( A \) and j-th row.
Fault Tolerant QR - decomposition Algorithm Based on Householder’s method

(j = 1,...,M for M ≤ N; j =1,...,N for M > N) of the resulting matrix R and PCS, QSC are the (N+1)-th and (N+2)-th columns (the columns of weighed checksums) of input matrix A respectively.

The macronodes (or simply nodes) G1 are located in vertices of the two-dimensional integer lattice

\[ Q1 = \{ D = (i, g, z): 1 ≤ i ≤ K; \ z=1, 2; \text{if} \ z=1 \text{then} \ i ≤ g ≤ N+8; \text{if} \ z=2 \text{then} i+5 ≤ g ≤ N+10 \}. \]

It should be noted that the i-th layer of graph G1 corresponds to the i-th step of the algorithm and that the significance of value z corresponds to the number of phase (1 or 2) of the i-th algorithm step (see expression (38)). There are five kinds of nodes in G1. The nodes with coordinates (i, i, 1), (i, i+1, 1) and (i, N+9, 2), (i, N+10, 2) which are marked with solid boxes correspond to the calculation of weighted checksums PRS\(^i\), QRS\(^i\) for the i-th column of the matrix A\(^i\) and weighted checksums PCS\(^i\), QCS\(^i\) for the i-th row of the resulting matrix R respectively, as well as, to the derivation and correction of erroneous elements within these columns and rows. The nodes with coordinates (i, i+2, 1), (i, i+3, 1) and (i, i+4, 1) which are marked with solid circles correspond respectively to the three times calculation of the i-th column norms and the significance of the elements \( \alpha^i, \beta^i, \gamma^i, a_{M+1}^i, a_{M+2}^i \) of the matrix A\(^i\). The nodes with coordinates (i, i+5, 1), ..., (i, N+6, 1) which are marked with circles correspond to the computation of the inner product \( y_k^i \) of i-th and k-th columns of matrix A\(^i\), while the nodes with coordinates (i, N+7, 1), (i, N+8, 1) which are marked with box correspond to the derivation and correction of erroneous elements within elements \( y_k^i \). The nodes with coordinates (i, i+5, 2), ..., (i, N+8, 2) which are marked with rectangles correspond to the computation of the elements \( a_{k+i+1}^i \) of the k-th (k=i+1,...,N+2) column of matrix A\(^i+1\) and to the derivation and correction of erroneous element within elements of erroneous column.

The data dependencies (or arcs) between nodes of G1 are represented by vectors

\[ d = [i \ g] \mathbf{^T}, \ d_1 = [1 \ 0] \mathbf{^T}, \ d_2 = [0 \ 1] \mathbf{^T}, \ d_3 = [1 \ -(N-i+3)] \mathbf{^T}, \text{and} \ d_4 = [1 \ -4] \mathbf{^T}. \]
where the first vector corresponds to passing variables $a_{jk}^i$ from the first sublayer ($z=1$) to the second sublayer ($z=2$) of $i$-th layer of the DG, i.e. at the second phase of $i$-th step of the algorithm and to passing variables $a_{jk}^{i+1}$ from the $i$-th layer to the $(i+1)$-st layer of the DG, i.e. at following step of the algorithm. The second vector corresponds to the pipeline propagation of the elements of the $i$-th (pivoting) column between the nodes (for $z=1$ and $z=2$). Besides this, vector $d_2$ also corresponds (for $z=2$) to the pipeline propagation of the elements of the erroneous, for example, $k$-th column $a_{jk}^i$ from node with coordinates $(i, k+4, 2)$ to the nodes with coordinates $(i, N+7, 2)$ and $(i, N+8, 2)$ for the error correction and calculation of the elements $a_{jk}^{i+1}$. The vector $d_3$ corresponds to the pipeline propagation of the elements of the $i$-th (pivoting) column and already corrected element $y_k^i$ from first ($z=1$) to second ($z=2$) sublayers of DG, while the vector $d_4$ corresponds the pipeline propagation the elements of $(i+1)$-th column at the following step of the algorithm.

From Fig.1 it is shown, that presence of global dependencies of various lengths between nodes of G1 given by vectors $d_3$ limits the set of permissible hardware solutions only non-effective ones. Therefore, to obtain such processor array structure with fixed number of PE’s which minimises both the number of PE’s performing I/O operations and amount of high-performance PE’s with division and square root capabilities, we used a purposive transformation of the DG G1. Firstly, the identical length for all vectors $d_3$ was derived. Secondly, the nodes with coordinates $(i, i, 1), ...,(i, i+4, 1)$ was joined into nodes with coordinates $(i, i, 1)$, while the nodes with coordinates $(i, N+7, 1),(i, N+8, 1)$ and $(i, N+7, 2),...(i, N+10, 2)$ was joined into nodes with coordinates $(i, N+7, 1)$ and $(i, N+7, 2)$ respectively. Note that mention nodes are destined for realisation in the high performance PE’s. Besides, the „empty” nodes were included into DG for pipeline propagation of the input data into PE’s. Thus, included into graph G2 additional layer minimises the amount of external inputs of DG. As a result, DG G2 was obtained (see Fig.2,a for case $M=N=5$).

The $i$-th layer of graph G2 analogously to the DG G1 corresponds to the $i$-th step of the algorithm and the significance of value $z$ corresponds to the number of phase (1 or 2) of the $i$-th algorithm step (see expression (38)).

The nodes of G2 are located in vertices of the two-dimensional integer lattice

$$Q2 = \{D = (i, g, z): 1 \leq i \leq K; z=1, 2; 1 \leq g \leq N+3\}.$$

There are five kinds of nodes in G2. The nodes with coordinates $(i, 1, 1)$ and $(i, N+3, 2)$ which are marked with solid rectangles correspond to the calculation of weighted checksums $PRS_i$, $QRS_i$ for the $i$-th column of the matrix $A_i$ and weighted checksums $PCS_i$, $QCS_i$ for the $i$-th row of the resulting matrix $R$ respectively, as well as, to the derivation and correction of erroneous elements within these columns and rows. Besides, the nodes with coordinates $(i, 1, 1)$ correspond to three times calculation of the $i$-th column norms and the significance’s of the elements $\alpha^i$, $\beta^i$, $\nu_i$, $y_i$, $a_{M+1i}$, $a_{M+2i}$ of the matrix $A_i$. The nodes with coordinates $(i, 2, 1), ...,(i, N+2, 1)$ which are marked with circles correspond to the computation of the inner product $y_k^i$ of $i$-th and $k$-th columns of matrix $A_i$, while the nodes with coordinates $(i, N+3, 1)$ which are marked with box correspond to the derivation and correction of erroneous elements within elements $y_k^i$. The nodes with coordinates $(i, 2, 2), ...,(i, N+2, 2)$ which are marked with rectangles correspond to the computation of the elements $a_{jk}^{i+1}$ of the $k$-th $(k=i+1,...,N+2)$ column of matrix $A_i^{i+1}$ and to the derivation and correction of erroneous element within elements of erroneous column. Finally, the nodes marked with big solid circles correspond to „empty” nodes with no arithmetic operation.
Fault Tolerant QR - decomposition Algorithm Based on Householder’s method

The data dependencies (or arcs) between nodes of $G_2$ are represented by vectors

$$d = [ i \ g]^T; \quad d_1 = [1 0]^T, \quad d_2 = [0 1]^T, \quad d_3 = [1 -(N+2)]^T, \quad d_4 = [1 -1]^T,$$

where the first vector corresponds only to passing variables $a_{jk}^i$ from the first sublayer ($z=1$) to the second sublayer ($z=2$) of $i$-th layer of the DG. The second and third vectors of graph $G_2$ are completely corresponded to the vectors $d_2$ and $d_3$ from DG $G_1$, while the vector $d_4$ corresponds to the passing variables $a_{jk}^{i+1}$ from the $i$-th layer to the $(i+1)$-st layer of the DG, i.e. at following step of the algorithm.

When implementing the LPGS scheme of partitioning (or decomposition) for the graph $G_2$, we try to decompose it into a set of $s = \lceil N/n \rceil$ subgraphs with the "same" topology. Moreover, according to the scheme, their must not exist bi-directional data dependencies between subgraphs. As evident from Fig.2,a, this can be done if the graph $G_2$ is cut using a set of straight lines parallel to $i$-axis, as shown in Fig.2,a. These lines decompose the graph $G_2$ into $s$ regular subgraphs with width equal $n$ nodes each (see Fig.2,a). Then in accordance with the chosen scheme of partitioning, all the
Fault Tolerant QR - decomposition Algorithm Based on Householder’s method PACT

Subgraphs of a set should be mapped into one array with a fixed number n of PEs. This can be done only by projecting each subgraph onto g-axis. As a result, the array architecture shown in Fig. 2, b is derived, using our approach proposed in [27]. This architecture, which is provided with an internal RAM module M(i), i = 1, ..., n, features simple scheme of fully local communications and small number of I/O channels. The total time T of executing the proposed parallel version of ABFT Householder’s algorithm on this array and processor utilisation \( \eta \) are approximately equal respectively to

\[
T = \sum_{i=1}^{s} \left[ 2(M-n(i-1))N/n + (M-n(i-1))(n-1) \right]N
\]

time steps, or

\[
T = \frac{N^2}{n} + (n-1)\frac{N^2}{2}
\]
times steps for case M = N, and

\[
\eta = \frac{W}{(T \times n)},
\]

where n is the number of PEs in the array, s = N/n and W is the computational complexity of the algorithm (38) represented by DG G1, W \( \approx \) \( 2N^3/3 + 8,5N^2 \) operations.

Using these formulae we obtain the following significance of utilisation \( \eta \):

\[
\eta \approx 0,66.
\]

It should be noted that with increasing in parameter s the value of \( \eta \) is also increasing.

6 Conclusions.

In this paper, we have proposed the fault-tolerant QR-decomposition algorithm based on Householder’s method of reflections and modified weighted checksum method. The purpose was to detect, locate and correct the single calculation errors occurred due to transient hardware faults during computation. The proposed algorithm enables to correct a single error among elements of any column (or row) of an input matrix A(M,N) on any from K = N or \( K = (M-1) \) steps of algorithm implementation. Consequently, it is possible to correct up to \( K \) single errors during solving the whole decomposition task. This effect is obtained by increasing the computational complexity of the original Householder’s method on \( 8,5N^2 + O(N) \) multiply-add operations. However, the original dependence graph G1 of proposed algorithm is few suitable for effective parallel realisation. Therefore, to obtain such processor array structure with fixed number of PE’s which minimises both the number of PE’s performing I/O operations and amount of high performance PE’s with division and square root capabilities, we used a purposive transformation of the DG G1 to the graph G2. Finally, the fixed-size linear array architecture with fully local communications, low I/O requirements and high processor utilisation has been designed for the parallel implementation of the proposed fault tolerant version of Householder’s algorithm.

In future works, we plan to apply the modified weighted checksum method to other important numerical problems such as singular value decomposition and eigenvalues problem.

References.

Fault Tolerant QR - decomposition Algorithm Based on Householder’s method _______________ PACT ___