Systolic-Type Implementation of Matrix Computations Based on the Faddeev Algorithm

R. Wyrzykowski

Dept. Math. & Computer Science
Technical University of Czestochowa
Czestochowa, Poland 42-400

J. S. Kanevski, O. V. Maslennikov

Dept. Computer Science
Kiev Polytechnic Institute
Kiev, Ukraine 252-056

Abstract

This paper deals with the problem of enhancing the versatility of VLSI processor arrays without undue addition of hardware, time/control overhead, and software complexity. A promissing approach to this problem is based on matrix computations carried out through the Faddeev algorithm. In the paper, we design a fixed-size, linear array architecture with fully local communications and, rather, straightforward control requirements. This high-throughput, systolic-type architecture allows us to minimize both I/O requirements and the number of processing elements performing complicated operations like divisions. To derive the array from a formal description of the Faddeev algorithm based on Gaussian elimination with partial pivoting, we use purposive transformations of the basic dependence graph of the algorithm before its space-time mappings onto array architectures.

1 Introduction

Recent advantages in VLSI technology have stimulated research in application-specific architectures which are tailored to particular applications. Among these architectures are application-specific processor arrays, which can have a different degree of specialization [1]. Systolic-type arrays [1,2] are examples of such architectures. Using massive parallelism/pipelining, these VLSI processor networks exploit the regularity inherent in many algorithms to achieve high performance while keeping local communications and low I/O requirements.

One of the principal problems encountered in designing these arrays is that of providing a sufficiently general range of functionality without undue addition of hardware, time/control overhead, and software complexity. A promissing approach to this problem is based [3,4] on the generality of linear (or matrix algebra), a class of operations whiche arises in a wide variety of application areas, including signal and image processing, real-time control, modelling and simulation, etc.

In computational linear algebra, the Faddeev algorithm [4,5,6,7,8,9] is inherently versatile. It enables one to perform many matrix operations including addition, multiplication, inversion, LU-decomposition, solution of linear systems, etc.

Since the underlaying procedure to carry out the Faddeev algorithm is matrix triangularisation, any processor array performing the algorithm should be based [8] on an architecture which can realize triangularisation efficiently. The triangular systolic array developed by Kung and Gentleman [10] is a common platform for two-dimensional (2-D) systolic architectures [4, 6, 7, 8, 9], which perform the Faddeev algorithm using either Gaussian elimination with neighbour pivoting or orthogonal triangularisation. The second approach is numerically stable but it requires relatively complex processing elements (PEs). Some of them must perform square root operations, which are not easily implementable for real-time applications. On the other hand, Gaussian elimination architectures employ simple PEs. But Sorenson has shown [11] that the error bound for pairwise pivoting is much worse than that for Gaussian elimination with partial pivoting [3], as well as, for orthogonal triangularisation.

This paper deals with the design of linear processor arrays which perform the Faddeev algorithm using Gaussian elimination. Unlike 2-D architectures, these one-dimensional (1-D) arrays allow one [12], firstly, to minimize the amount of I/O channels because they are connected only with the first or/and the last PE. Secondly, a large structure can be constructed simply by concatenation of smaller arrays. Thirdly, a linear array requires memory bandwidth which is independent of the size of the array. In the paper, we employ partial pivoting instead of neighbour pivoting. When processing time for linear architectures is considered, neighbour pivoting has not any advantage over partial pivoting [13] being much less reliable.

The paper is organized as follows. At first (Section 2), we describe the chosen version of the Faddeev algorithm and some applications of it. The next section deals with a basic dependence graph of the algorithm. This graph is then used (Section 4) for the design of linear array architectures performing the algorithm. To derive arrays with desired features, some purposive transformations of the basic graph are employed before space-time mappings of graphs onto array architectures. Since the arrays derived in this way feature a strong dependence of their sizes upon sizes of matrices being processed, we show (Section 5) how these architectures should be modified in order to

0-8186-6322-7/94 $04.00 © 1994 IEEE

process large size matrices on fixed-size arrays. Section 6 provides conclusions.

2 Faddeev algorithm and its applications

Starting with $N \times N$, $N \times R$, $P \times N$ and $P \times R$ input matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$ and $\mathbf{D}$, respectively, the Faddeev algorithm is intended [5] for solving matrix equations of the type

$$\mathbf{X} = \mathbf{C}\mathbf{A}^{-1} + \mathbf{D} \quad (1)$$

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

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

The essence of the Faddeev algorithm [4, 5] consists in reducing the lower left quadrant of the matrix $\tilde{\mathbf{F}}$ (i.e. $\mathbf{C}$-matrix) to zero matrix, while in the lower right quadrant of the matrix $\tilde{\mathbf{F}}$ (i.e. in place of $\mathbf{D}$-matrix), the resultant $P \times R$ matrix $\mathbf{X}$ is formed. To carry out the above-stated operations with $\mathbf{A}$ being a nonsingular matrix, Gaussian elimination is used. Hence, in the course of computations, the joint matrix $\tilde{\mathbf{F}}$ is being transformed into the following matrix:

$$\begin{bmatrix} \mathbf{R} & \mathbf{B}^* \\ \mathbf{0} & \mathbf{X} \end{bmatrix}$$

where $\mathbf{R}$ is the upper triangular matrix. A considerable degree of versatility of the Faddeev algorithm stems from the fact that Eq. 1 allows us to solve a set of problems. Some of them are listed below:

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

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

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

There are [7, 8] other important modifications of the Faddeev algorithm. As a result, it can be employed, for example, in fast solving of linear programming problems using the Karmarkar algorithm.

To provide numerical stability of the Faddeev algorithm, we employ Gaussian elimination with partial pivoting within columns [3, 13]. As a result, at the i-th step ($i = 1, \ldots, N$) of the algorithm, the elimination of elements f^i_{ji} ($j = i+1, \ldots, N+P$), which belong either to the original matrix $\tilde{\mathbf{F}} = \mathbf{F}^1$ (for $i = 1$) or to the partially transformed matrix $\tilde{\mathbf{F}}^i$ (for $i > 1$), is preceeded by successive comparisons of f^i_{ji} ($j = i+1, \ldots, N$) with the pivot element f^i_{ii}. If

$$| f^i_{ji} | > | f^i_{ii} |$$

then the i-th and j-th rows of the matrix $\mathbf{F}^i$ are interchanged, and a boolean variable v_{ji} is set to 1. In the opposite case, the row interchange does not take place, and v_{ji} is set to 0.

After completing all comparisons and interchanges for a given step, the pivot row $\tilde{f}^i_i = [f^i_{ik}]$ with the pivot element f^i_{ii} is finally derived, where $k = i, \ldots, N+R$. Then the elimination of elements f^i_{ji} ($j = i+1, \ldots, N+P$) starts. It is accompanied by transformations of rows of the matrix $\tilde{\mathbf{F}}$, from the $(i+1)$-st row to the $(N+P)$-th row. These transformations consist in the element-by-element summation of each row with the pivot row, which is in advanced multiplied by coefficient $m_{ji} = -f^i_{ji}/f^i_{ii}$.

Thus, to provide a correct realization of the algorithm, the selection of pivot elements as well as corresponding interchanges are limited only to the upper (corresponding to the matrices $\mathbf{A}$ and $\mathbf{B}$) quadrants of matrices $\tilde{\mathbf{F}}^i$. However, the elimination process is carried out within all quadrants of $\tilde{\mathbf{F}}^i$. Naturally, in the N-th step, the element f^N_{NN} is immediately taken as a pivot, without any comparison.

The described-above version of the Faddeev algorithm can be expressed in the following form:

```
for i:= 1 step 1 until N do
  {selection of the pivot element
  within the i-th column}
  for j:= i+1 step 1 until N do
  begin
    if abs(f[i,i]) < abs(f[j,i]) then
    begin
      s:=f[i,i]; f[i,i]:=f[j,i]; f[j,i]:=s;
      v[j,i]:= true;
    end
        else v[j,i]:= false;
    {row interchanges}
    for k:= i+1 step 1 until N+R do
      if v[j,i] = true then
      begin
       s:=f[i,k]; f[i,k]:=f[j,k]; f[j,k]:=s;
      end;
  end j;
  {calculation of multipliers m[j,i]}
  for j:= i+1 step 1 until N+P do
  if f[i,i] = 0 then m[j,i]:= 0
    else m[j,i]:= -f[j,i]/f[i,i];
  {transformations of rows of the
   joint matrix, from the (i+1)-st
   row to the (N+P)-th one}
  for j:= i+1 step 1 until N+P do
    for k:= i+1 step 1 until N+R do
    f[j,k]:= f[j,k] + m{j,i]*f[i,k]
end i                                          (2)
```

3 Basic dependence graph of the Faddeev algorithm

In spite of using **if** ... **then** ... **else** statements in algorithm (2), the order of execution of its operators is unambigously determined before computations. This allows us to construct [1] its dependence graph (DG) called the basic DG and denoted by $\mathbf{G}_B$. It corresponds to the execution of algorithm (2) in accordance with the given lexicographical order. Nodes of $\mathbf{G}_B$ are distributed in nodes of the three-dimensional lattice $Q_B = \{\bar{K} = (i,j,k) : 1 \leq i \leq N,\ i+1 \leq j \leq N+P,\ i \leq k \leq N+R\}$. This lattice can be visualized as a truncated pyramid possessing a rectangular base with the size of $(N+R) \times (N+P-1)$ nodes. The height of the lattice is N units (or layers).

The graph $\mathbf{G}_B$ is shown in Fig. 1, where $N=4$, $R=P=1$. It should be noted that the i-th layer of $\mathbf{G}_B$ $(i=1,\ldots,N-1)$ is composed of two sublayers for which we assume $z=1$ or $z=2$. We will call these two sublayers pivot or elimination sublayer, respectively. The first sublayer with $z=1$ consists of $(N-i) \times (N+R-i+1)$ subnodes, and corresponds to the selection of the pivot element within the i-th column of the matrix $\mathbf{F}^i$, as well as, to the described-above interchanges of its rows, from the i-th row to the N-th row. These interchanges are carried out under the control of boolean variables v_{ji} generated during the selection process, where $j=i+1,\ldots,N$. The second sublayer with $z=2$ consists of $(N+P-i)\times(N+R-i+1)$ subnodes, and corresponds to the computation of coefficients m_{ji}, where $j=i+1,\ldots,N+P$, followed by transformations of rows of the joint matrix, from the $(i+1)$-st row to the $(N+P)$-th row. The highest, N-th layer of $\mathbf{G}_B$ is composed of only the elimination sublayer with $P \times (R+1)$ subnodes.

The data dependencies (or arcs) between nodes of the graph $\mathbf{G}_B$ are represented by the following vectors: $\mathbf{d}_1=[100]^t$, $\mathbf{d}_2=[010]^t$, $\mathbf{d}_3=[001]^t$, $\mathbf{d}_4=[110]^t$ and $\mathbf{d}_5=[0\ i-N\ 0]^t$. Note that $\mathbf{d}_1$ and $\mathbf{d}_4$ correspond to the passing of variables f^i_{jk} and f^i_{ik}, respectively, from the elimination sublayer of the previous, $(i-1)$-st layer $(i \neq 1)$ to either the pivot sublayer of the actual, i-th layer, if $j \leq N$, or to the elimination sublayer of this layer, if $j > N$. For $i=1$, the elements of the input matrix $\tilde{\mathbf{F}}$ are fed into pivot subnodes of the first layer. The selection of the pivot row at the pivot sublayer of a layer, as well as the pipeline propagation of this row between subnodes of the elimination sublayer of the same layer, are described by the vector $\mathbf{d}_2$. The vector $\mathbf{d}_3$ corresponds to the pipeline propagation of boolean variables v_{ji} or coefficients m_{ji} between subnodes of a pivot or elimination layer, respectively.

We take particular note of the vector $\mathbf{d}_5$. It corresponds to the transfer of the pivot row $\bar{f}^i_i$, finally determined in the pivot sublayer of the i-th layer, to the elimination sublayer of the same layer. While the rest of vectors correspond to local data dependencies between nodes of the graph $\mathbf{G}_B$, the vector $\mathbf{d}_5$ introduces global dependencies in the graph. Besides the above-described data dependencies between nodes of $\mathbf{G}_1$, there are also dependencies inside $(N-i)\times(N+R-i+1)$ nodes of the i-th layer. Each of these dependencies originates in the pivot subnode of a node and ends in the elimination subnode of the chosen node. They are produced by passing results of row interchanges, except for a pivot row, from a pivot sublayer to the elimination sublayer.

4 Design of linear arrays for the Faddeev algorithm

The DG of an algorithm is known [15] to reflect all essential properties of dependencies between its operators. That is why, the above form of parallel algorithm representation is used as a basis in many methods (e.g. [1, 2, 12, 14]) for the design of processor arrays. According to these methods, the DG is mapped onto structural schemes $\mathbf{C} =< S,T,\Phi >$ of processor arrays performing the given algorithm, where S is the directed graph called the array structure, T is the synchronization function, and Φ is the set of PE operation algorithms. For this purpose, the DG, which is determined in an integer lattice, is subject to a set of monotonic and injective mappings $\mathbf{F}$. Each of these mappings, which are usually linear functions, contains the processor assignment F_S and the schedule mapping F_T, where F_S determines a structure S, F_T determines a function T, and both of them determine a set Φ.

However, these methods can not be regarded as complete ones. For example, they neglect a possibility to generate not a single variant of DG but a set of them. At the same time, any value of space-time mappings must lie within the domain of such parallel implementations of the algorithm which are allowed by a certain version of its DGs. This version not only determines permissible topologies of interprocessor connections, but also partly predetermines the execution order for operators of the algorithm. Hence, to obtain a wider set of structural schemes performing the given algorithm, it is desirable to expand this domain by generating a variety of DGs for the initial description of the algorithm. Such a generation, which is equivalent to transformations of the basic DG, allows us to unfold the inherent properties of the algorithm in order to use them for deriving array architectures with desired features.

4.1 Graph transformations in the design of arrays for the Faddeev algorithm

In the graph $\mathbf{G}_B$, the critical paths (paths with a maximum length) have the length of $N(N+5)/2 + R+P-2$ subnodes. It gives the lower bound for the total computation time t^* required by the algorithm to process a particular matrix $\tilde{\mathbf{F}}$. Consequently, when matrices are processed individually, all 2-D processor arrays will manifest the low processor utilization of $\eta^* = W/(t^*\tilde{M}) \approx 0(N^{-1})$. Here $W \approx 0(N^3)$ or $\tilde{M} \approx 0(N^2)$ is the number of subnodes in the graph $\mathbf{G}_B$ or PEs in a 2-D structure, respectively.

Before passing on to the design process for 1-D (or linear) arrays, we note that the presence of global dependencies in $\mathbf{G}_B$ limits the set of array structures S with fully local communications between PEs. Indeed,

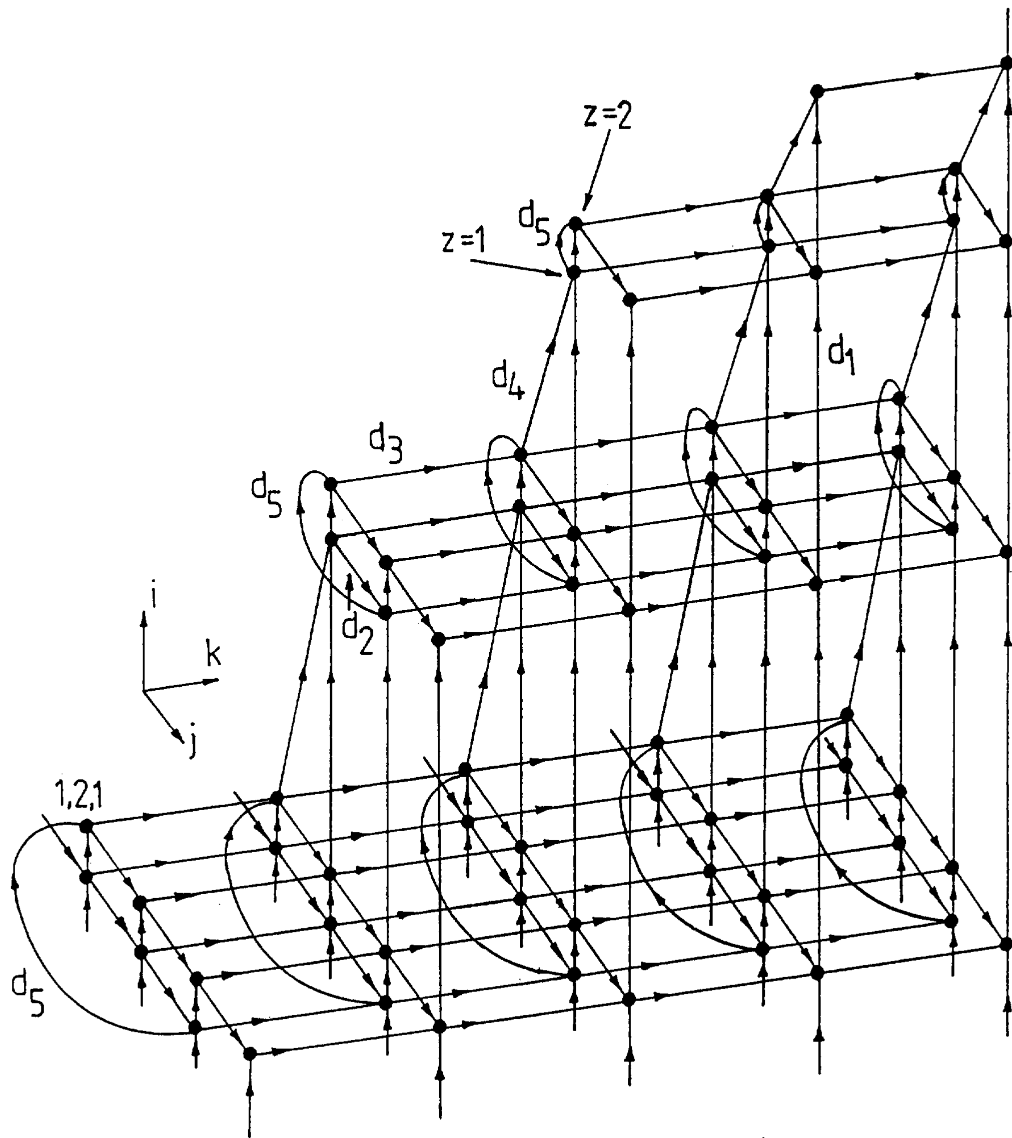

Figure 1: **Basic DG for algorithm (2).**

for 2-D structures, only the projection of the graph along the vector $\mathbf{r}^* = [0\ 1\ 0]$ satisfies this condition. Hence, to simplify the design of 1-D array structures, we transform the three-dimensional graph $\mathbf{G}_B$ into a 2-D graph $\mathbf{G}_1^*$, by projecting $\mathbf{G}_B$ along the vector $\mathbf{r}^*$. As a result, all nodes lying at a straight line parallel to $\mathbf{r}^*$ merge into a single macronode, which represents a macrooperation performed on an entire column of $\mathbf{F}^i$. We again transform the graph $\mathbf{G}_1^*$ by composing the pivot and elimination sublayers of the i-th layer of $\mathbf{G}_1^*$ into one layer, where $i = 1, \ldots, N-1$. Having done this, we get the graph $\mathbf{G}_1$. It is shown in Fig. 2a, where $N = 6$, $R = 3$. The diagonal macronodes with coordinates (i, i), where $i = 1, \ldots, N$, correspond to the selection of pivots and generation of boolean variables v_{ji}, $j = i+1, \ldots, N$, as well as, to the computation of coefficients m_{ji}, where $j = i+1, \ldots, N+P$. An off-diagonal macronode with coordinates (i, k), where $k = i+1, \ldots, N$, corresponds to interchanges of elements within the k-th column of the matrix $\mathbf{F}^i$, and subsequent calculations of elements of the k-th column of $\mathbf{F}^{i+1}$.

A set of 1-D structures S with fully local interconnections correspond to the 2-D graph $\mathbf{G}_1$. One of them is the structure S_1, which is shown in Fig. 2b, where $N = 6$, $R = 3$. This structure, which corresponds to the projection of $\mathbf{G}_1$ along i-axis, contains N PEs of the first type, i.e. having a division unit in addition to a multiplication-addition unit, and R PEs of the second type, i.e. without a division unit.

The drawbacks of the structure S_1 are comparatively large numbers of PEs and I/O channels, as well as, the presence of N PEs containing division units. These shortcomings can be eliminated by projecting the graph $\mathbf{G}_1$ along the vector $\mathbf{r} = (1, 1)$. As a result, we get the structure S_2, which is shown in Fig. 2c, where $N = 6$, $R = 3$. While having the same number of PEs as S_1, the structure S_2 has only one PE with a division unit. Another property of S_2 is the presence of data links which originate in the s-th PE and ends in the $(s-1)$-th PE, where $s = 2, \ldots, N+R$. These links are responsible for the interprocessor transfer of variables f_{jk}^{i+1}, which are calculated at the i-th step of the algorithm. Moreover, after executing the i-th step, the $(N+R-i+1)$-th PE transfers their intermediate results to the $(N+R-i)$-th PE, and then ceases to participate in computations.

In view of practical realization, the structure S_2 still suffers from a number of shortcomings, the most significant of which is a large number of I/O channels. This drawback can be easily avoided if we project the graph G_1 along k-axis. This projection results in the structure S_3, which has only N PEs (see Fig. 3d). Moreover, only the first and the last PEs of this structure perform I/O operations. Manifesting these advantages, the structure S_3 suffers, however, from the limitation that all its PEs must perform divisions in addition to multiply-add operations.

To obtain such a structure S which, while retaining the above-stated advantages of S_3, minimizes the number of PEs containing a division unit, we try to transform the triangular part of the graph $\mathbf{G}_1$, with the rectangular part being fixed. For this purpose, we place all diagonal macronodes of the graph $\mathbf{G}_1$ along k-axis, and then redraw the graph $\mathbf{G}_1$ preserving all interconnections between its macronodes. This transformation can be thought as a rotation of the triangular part of $\mathbf{G}_1$ by an angle of 45° clockwise. As a result, coordinates (i, k) of macronodes of the triangular part, where $i = 1, \ldots, N$, $k = i, \ldots, N$, are changed according to the following formulae: $k^* = k, i^* = N-k+i$. In this way, we derive an intermediate graph $\mathbf{G}_2^*$, which is shown ($N = 6, R = 3$) on the left-hand of Fig.3a, while the structure S_4, which corresponds to the projection of $\mathbf{G}_2$ along k^*-axis, is presented on the right-hand. Therefore, we now have a structure which consists of N PEs; only its last PE contains a division unit. However, the number of I/O channels is still very large, and it depends on N.

In order to reduce this number to one, we complete the graph $\mathbf{G}_2^*$ with "empty" macronodes, which provide the input of matrix $\tilde{\mathbf{F}}$ in accordance with Fig.3b, where the resulting graph $\mathbf{G}_2$ is depicted. After projecting it along k^*-axis, we obtain the structure S_5 shown on the right-hand of Fig. 3b. Since we have restricted the input of $\tilde{\mathbf{F}}$-matrix to only the border macronodes of $\mathbf{G}_2$, this matrix is fed into the structure S_5 by means of a single input channel. Lastly, to complete the design of a structural scheme $\mathbf{C}_5$ which corresponds to both the graph $\mathbf{G}_2$ and structure S_5, we derive a schedule mapping F_T^5, using the generalized mapping methodology [1]. Aiming at the minimization of the algorithm exection time and assuming only linear mappings, we obtain the following schedule:

$$F_T^5(\bar{K}^*, z) = (N+P-1)i^* + j + (N+P)k^* + (N-1)z + c$$

Here $\bar{K}^* \in Q^*$, and

$$Q^* = \{\bar{K}^* = (i^*, j, k^*) : 1 \le i^* \le N,\ 1 \le j \le N+P,\ 1 \le k^* \le N+R\},\ z \in Z = \{1, 2\}$$

and c is a constant chosen in such a way that

$$t_{min} = \min_{\bar{K}^* \in Q^*, z \in Z} F_T^5(\bar{K}^*, z) = F_T^5([1\ 1\ 1], 1) = 1$$

As a result, we have $c = -3N - 2P + 2$.

The scheme $\mathbf{C}_5$ is a linear array with N PEs; only the last PE contains a division unit, while the rest of PEs, which are of the same type, are not provided with it. The internal structures of the k^*-th ($k^* \neq N$) and the N-th PEs are detailed in Fig.4a and Fig. 4b, respectively, where D(N-1), D(N+P) and D(P) are delay lines (or FIFO-buffers) with corresponding lengths, M denote multiplexers, R denote registers, C is a comparator, and DV is a division unit. In the buffer D(N+P), the width of its words is sufficient to store coefficients m_{ji} and boolean variables v_{ji}.

When matrices are processed individually, the structural scheme synthesized above manifests the total execution time of

$$t_5^* = t_{max} - t_{min} + 1 = t_{max}$$

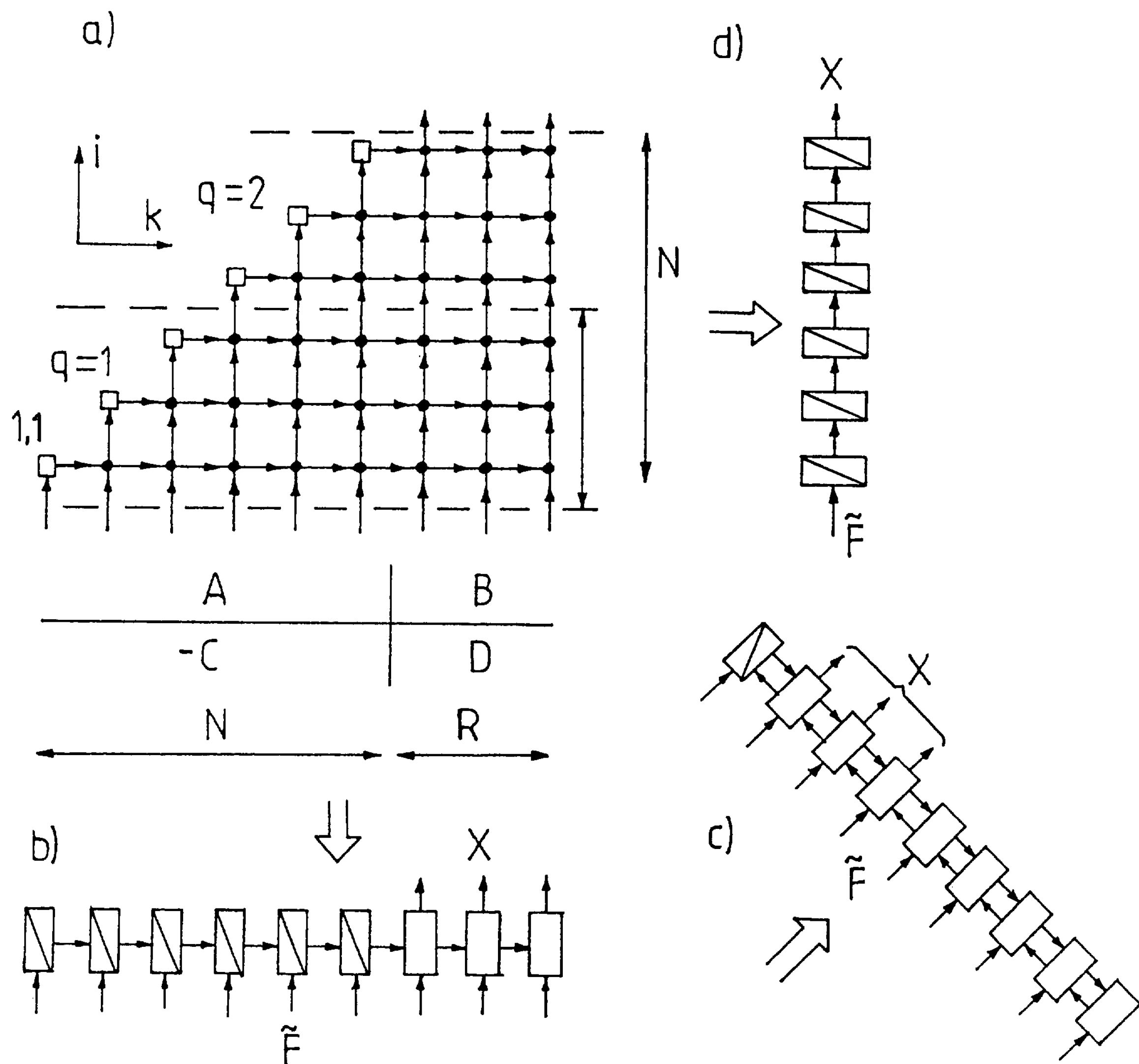

Figure 2: **Design of linear arrays based on the basic DG: (a) basic DG after reducing its dimension; (b) structure S_1; (c) structure S_2; (d) structure S_3.**

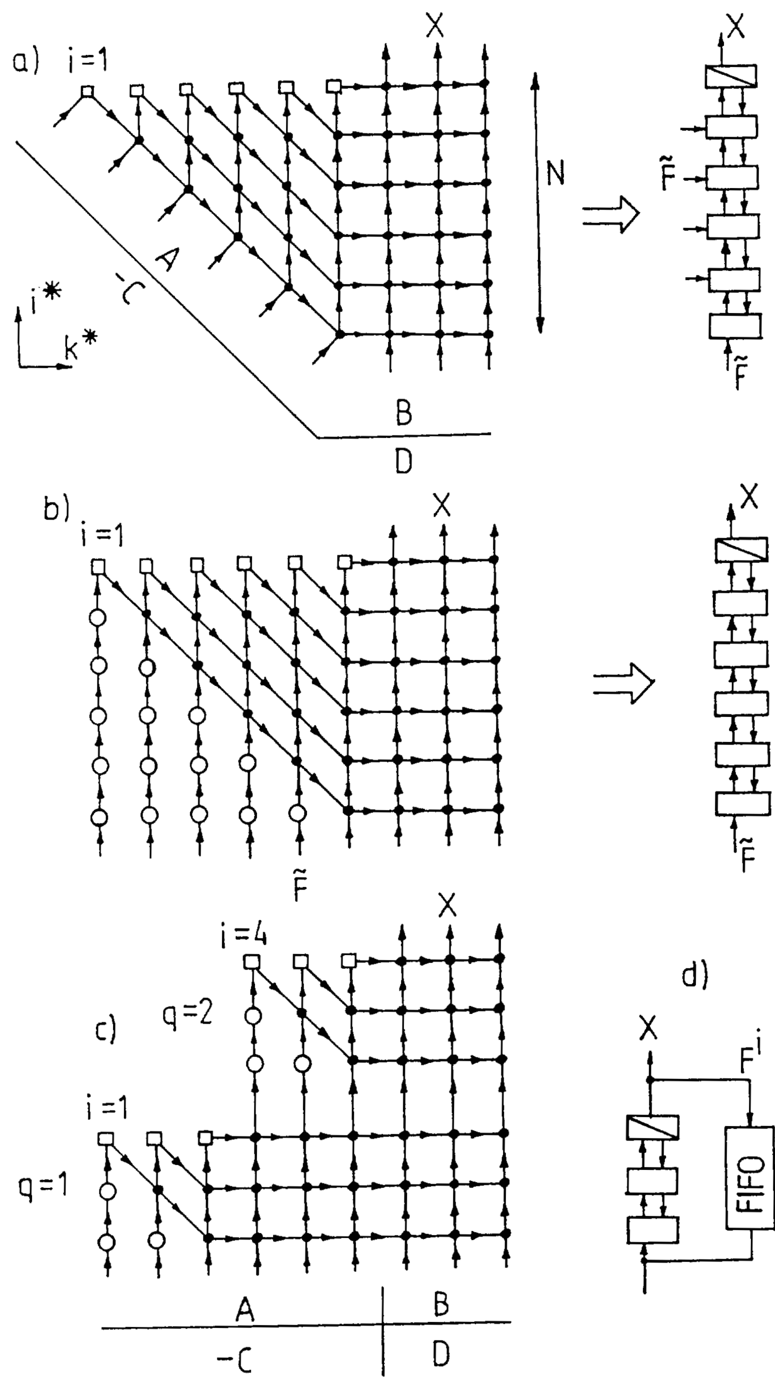

Figure 3: **Design of linear arrays, using transformations of the basic DG: (a) intermediate DG and structure S_4 corresponding to it; (b) resulting DG and structure S_5 corresponding to it; (c) DG partitionning for the LPGS scheme; (d) fixed-size linear array for the Faddeev algorithm.**

time steps, where

$$t_{max} = \max_{\bar{K}^* \in Q^*, z \in Z} F_T^5(\bar{K}^*, z) = F_T^5([N \;\; N+P \; N+R], \;\; z=2)$$

so that

$$t_5^* = (N+R-1)(N+P)+(N+P-1)N+N$$

steps. The processor utilization can be estimated as $\eta_5^* = W/(t_5^* \tilde{M})$, where $\tilde{M} = N$ is the number of PEs in the array, and W is the computational complexity of algorithm (2). Since W is given by $N^3/3+N^2(P+R)/2+NPR$ divisions and multiply-add operations, we have $\eta^* \approx 0.39$ for $N=P=R$. To improve the processor utilization, a number of matrices need to be processed sequentially. Because a new matrix $\tilde{\mathbf{F}}$ can be processed as soon as the input of the previous matrix $\tilde{\mathbf{F}}$ is completed, the above scheme is characterized by the block pipelinig period [1] of $t_5^{\cdot} = (N+P)(N+R)$ steps and the average processor utilization of $\eta_5^{\cdot} = W/(t_5^{\cdot} \times \tilde{M}) \approx 0.58$ for $N=P=R$. Note that with an increase in parameters P or/and R, the processor utilization also increases. For example, when $P=R=2N$ or $N=P$, $R=4N$, we have $\eta_5^* \approx 0.53$, $\eta_5^{\cdot} \approx 0.7$ or $\eta_5^* \approx 0.57$, $\eta_5^{\cdot} \approx 0.68$.

4.2 Design of fixed-size linear array

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 [2]. To provide this ability, two partitioning methods [1] 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 DG into a set of regular subgraphs, but differ in the way how these subgraphs are mapped onto resulting structural schemes. In the LSGP 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 can be stored in buffers outside the processor array. We employ this scheme in order to implement the Faddeev algorithm on a linear array with $n < N$ PEs, where n is a fixed number. Starting with the graph $\mathbf{G}_1$, we try to decompose it into a set of $s =]N/n[$ subgraphs having the "same" topology, where $]x[$ denotes the nearest integer equal to or greater then x. As evident from Fig. 2a, this can be done only if we "cut" the graph $\mathbf{G}_1$ using a set of straight lines parallel to k-axis. These lines decompose the graph $\mathbf{G}_1$ into regular subgraphs G_1^q with n layers each, where $q = 1, \ldots, s$. Then, the above-described (Subsection 4.1) rotation of the triangular part of $\mathbf{G}_1$ by an angle of 45° is individually used for every $\mathbf{G}_1^q$. Lastly, after completing each of subgraphs with "empty" macronodes, a set of s subgraphs $\mathbf{G}_2^q$ with the "same" topology is obtained (see Fig. 3c). Using the LPGS method and taking into account the way how the structural scheme $\mathbf{C}_5$ has been derived from the graph $\mathbf{G}_2$, we come to the conclusion that by providing this scheme with an external FIFO buffer for storing and recirculating the intermediate data, a fixed-size processor arrays for the Faddeev algorithm can be obtained (see Fig. 3d).

For the possible variants of implementing the array, it will manifest different performance characteristics. The simplest variant assumes that lengths of all FIFO buffers are constant during the execution of the algorithm. In this case, FIFO buffers within PEs have lengths of $N-1$, $N+P$ and P cells, respectively, while the external buffer contains $L_{ex} = (N+P)(N+R)-c^*$ cells, where $c^* = n(2(N+P)-1)-P+1$. Executing algorithm (2), the fixed-size processor array has now to perform a computational work which corresponds to the volume of the solid shown in Fig. 5a. As a result, the total execution time is maximum. It is given by the following expression:

$$t_{5,1}^* = s(N+P)(N+R) + (N+P-1)(n-1)+(N-1) \quad (3)$$

Here the first component takes into account the time interval required by the input of both the original matrix $\tilde{\mathbf{F}} = \mathbf{F}^1$ and intermediate matrices $\mathbf{F}^r$, where $r = n, 2n, \ldots, (s-1)n$. The rest of the components correspond to the subsequent completion of computing the resultant matrix $\mathbf{F}^{N+1}$. Note that all intermediate matrices $\mathbf{F}^r$ are extended to $(N+P)\times(N+P)$ matrices by introducing "dummy" elements.

To reduce the execution time without having to control lengths of FIFO buffers within PEs, only the length L_{ex} of the external buffer can be varied in the course of computations. In this case, for storing elements of a matrix $\mathbf{F}^r$ which are generated as a result of processing the q-th subgraph, where $r = q \times n, q = 1, \ldots, s-1$, we should provide $L_{ex}^q = (N+P) \times (N+R-n(q-1)) - c^*$ cells. Executing the Faddeev algorithm, the processor array will now perform a computational work which corresponds to the volume of the solid depicted in Fig.5b. Hence, the total execution time will be given by

$$t_{5,2}^* = \{\sum_{q=1}^{s}(N+R-n(q-1))(N+P)\} + (N+P-1)(n-1)+(N-1) \quad (4)$$

where the components have the same meaning as before. Note only that all intermediate matrices $\mathbf{F}^r$ are now extended to $(N+R-n(q-1)) \times (N+P)$ matrices. After transforming Eq. 4, we obtain finally

$$t_{5,2}^* = s(N+P)(N+R) - sn(s-1)(N+P)(N-n)/2 + (N+P-1)(n-1)+(N-1) \quad (5)$$

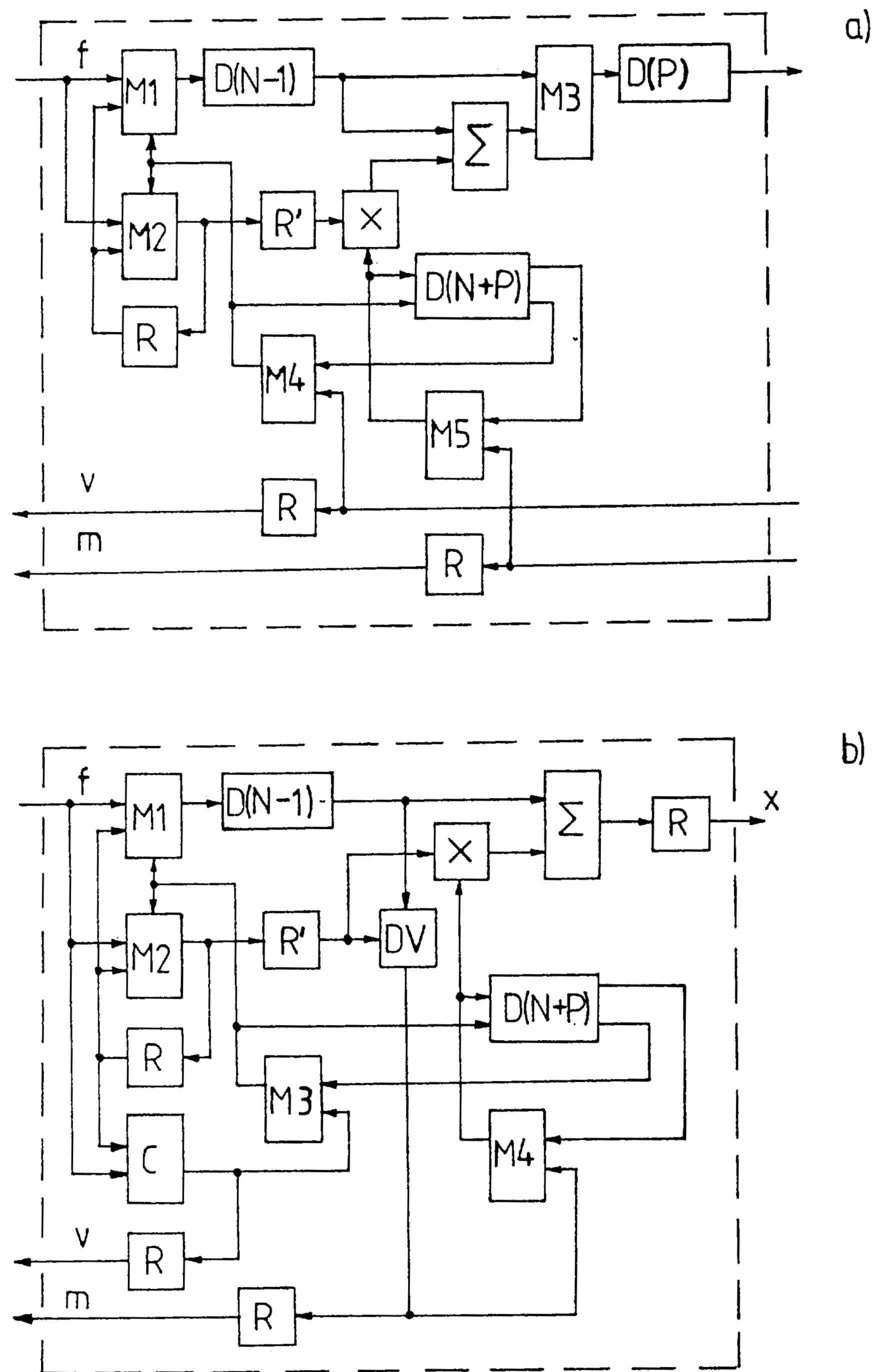

Figure 4: **Internal structures of PEs: (a) k−th PE, where $k = 1, \ldots, N-1$; (b) N−th PE.**

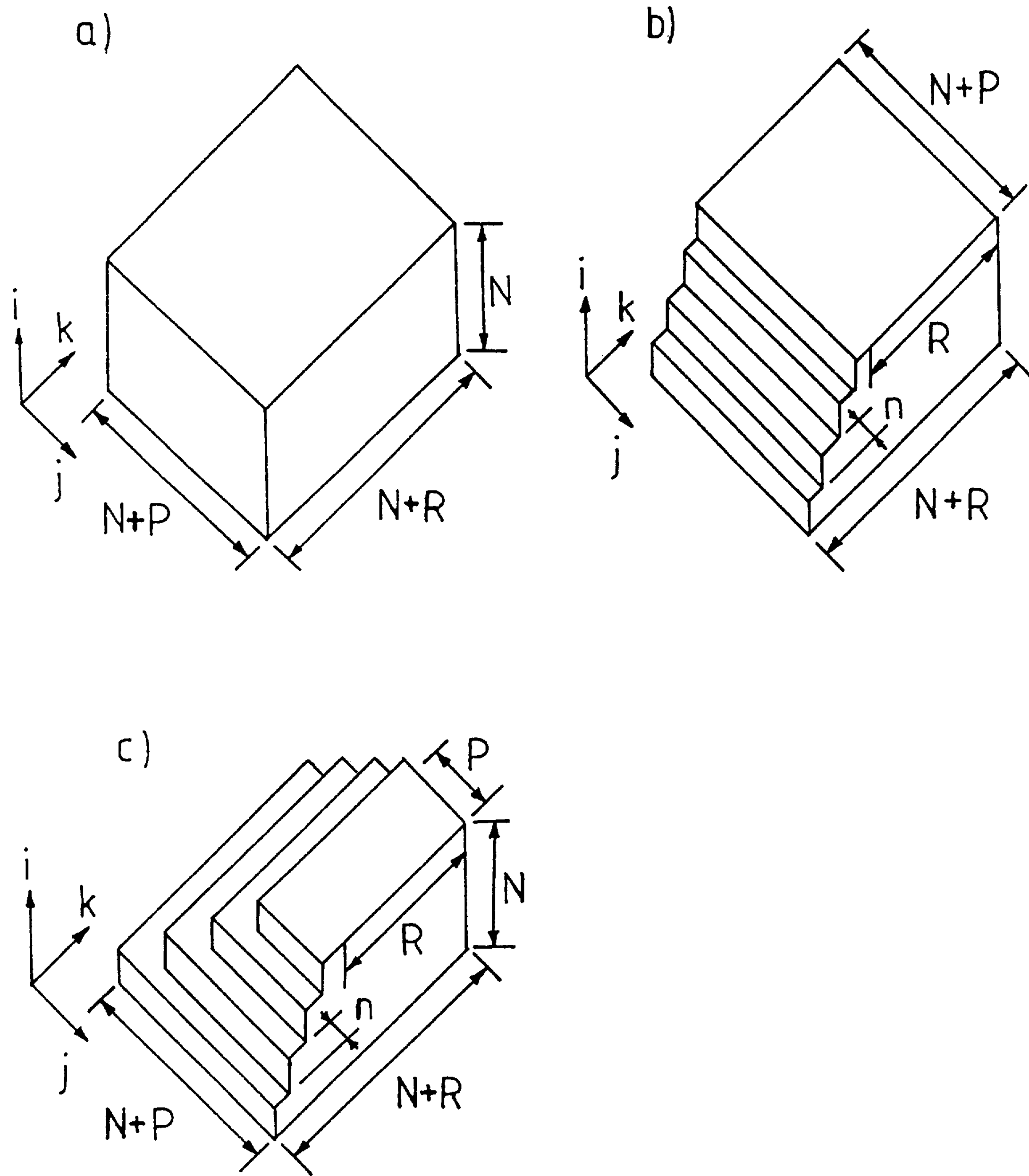

Figure 5: **Computational work corresponding to different variants of implementing the fixed-size array: (a) lengths of all FIFO buffers are constant; (b) only the length of the external buffer are varied; (c) lengths of both the external buffer and buffers within PEs are varied.**

Lastly, varying lengths of all FIFO buffers, we can achieve a further decrease in the total execution time. In this case, after executing the q-th subgraph, those internal FIFO buffers which are denoted by D(N+P) and D(N - 1) in Fig. 4 should contain $N+P-n(q-1)$ and $N-n(q-1)$ cells, respectively, while for the external buffer we should provide

$$L_{ex}^{q} = (N+P-(q-1)n)(N+R-(q-1)n) - c^{\#}$$

cells, where

$$c^{\#} = n\{2(N+P-(q-1)n)-1\} + P - 1$$

The processor array will now perform a work which is given by the volume of the solid presented in Fig. 5c. As a result, the total execution time is minimum. It can be expressed by the following formula:

$$\begin{aligned} t_{5,3}^{*} &= \sum_{q=1}^{s}(N+R-n(q-1))(N+P-n(q-1)) \\ &\quad + (N+P-n(q-1)-1)(n-1) \\ &\quad + (N-n(q-1)-1) \\ &\quad + (s-1)(n(n-1)-1) \qquad (6) \end{aligned}$$

All but the last components of Eq. 6 have the same meaning as before. The presence of the last component reflects the fact that the loading of an intermediate matrix $\mathbf{F}^{r}$ can start only $n(n-1)-1$ steps after the processing of the previous matrix $\mathbf{F}^{r-1}$ is completed. The final form of Eq. 6 is as follows:

$$\begin{aligned} t_{5,3}^{*} &= s(N+P)(N+R) - sn(s-1)(2N+P+R) \\ &\quad + n^{2}s(s-1)(2s-1)/6 \\ &\quad + n(N+P-n(q-1)-1) \\ &\quad + (s-1)(n(n-1)-1) \qquad (7) \end{aligned}$$

For small values of the parameter $s =]N/n[$, the implementation of the above-described variants results in small differences in the total execution time. For example, if $s=2$ and $N=P=R$, then $t_{5,1}^{*} \approx 8N^{2}$, $t_{5,2}^{*} \approx 7N^{2}$ and $t_{5,3}^{*} \approx 6.33N^{2}$. But when $N \gg n$, these differences become more essential. Indeed, for $N \bmod n = 0$, Eqs. 3, 5 and 7 can be reduced to the following formulae:

$$\begin{aligned} t_{5,1}^{*} &\approx s(N+P)(N+R) \\ t_{5,2}^{*} &\approx s(N+P)(N+R) - (s-1)(N+P)N/2, \\ t_{5,3}^{*} &\approx s(N+P)(N+R) - \\ &\quad (s-1)(2N+P+R)N/2 + (s-1)N^{2}/3 \end{aligned}$$

Using these formulae and supposing as before that $N=P=R$, we obtain $t_{5,1}^{*} \approx 40N^{2}$, $t_{5,2}^{*} \approx 31n^{2}$, $t_{5,3}^{*} \approx 25N^{2}$ for $s=10$, and $t_{5,1}^{*} \approx 400N^{2}$, $t_{5,2}^{*} \approx 301 \times N^{2}$, $t_{5,3}^{*} \approx 235N^{2}$ for $s=100$. Finally, when $s \rightarrow \infty$, we have $t_{5,1}^{*} \approx (12/7)t_{5,3}^{*}$, $t_{5,2}^{*} \approx (9/7)t_{5,3}^{*}$, and $\eta_{5,1}^{*} \approx 7/12 \approx 0.58$, $\eta_{5,2}^{*} \approx 7/9 \approx 0.78$, $t_{5,3}^{*} \approx 1$.

Therefore, the third variant of implementing the fixed-array allows us to optimize both the total execution time and processor utilization. From this point of view, the second variant is not so efficient, but unlike the third variant it does not require to vary lengts of FIFO buffers within PEs. This advantage considerably simplifies a practical realization of the array.

5 Conclusions

One ot the principal problems encountered in the design of VLSI processor arrays is that of providing a sufficiently general range of functionality without undue addition of hardware, time/control overhead, and software complexity. A promising approach to this problem is based on implementation of the Faddeev algorithm for computing the matrix expression $\mathbf{X} = \mathbf{C}\mathbf{A}^{-1}\mathbf{B} + \mathbf{D}$, where $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ and $\mathbf{D}$ are $N \times N$, $N \times R$, $P \times N$ and $P \times R$ matrices, respectively.

In this paper, we have designed a new architecture of linear array performing the Faddeev algorithm based on Gaussian elimination with partial pivoting. Unlike 2-D (or planar) architectures, this 1-D array allows us to minimize the amount of I/O channels because these channels are connected only with the first and the last PEs of the array. To derive it, some purposive transformations of the basic dependence graph of the algorithm have been used before the space-time mapping of the graph onto the architecture.

Another important feature of the resulting array, which contains N PEs with FIFO buffers, is that all the complicated operations like divisions are carried out by the same boundary PE, whereas other PEs perform multiplication-additions. This Faddeev-based array architecture manifests a simple scheme of fully local communications, as well as, the block pipelining period of $(N+P)(N+R)$ time steps. This value is minimal for a computing device with a single communication channel performing the input of matrices A, B, C and D.

For some computations, the Faddeev-based approach will not lead to the most efficient implementations, but the enhanced versatility should more than compensate for this defficiency. For example, to provide the ability to process large size matrices on a fixed-size array, the uniform and simple scheme for the partitioning of the original dependence graph into regular subgraphs could be employed. These subgraphs have been then mapped onto the resulting architecture in accordance with the locally parallel globally sequential method. As a result, by providing the original linear array containing n PEs ($n = const$) with a single FIFO buffer, the fixed-size, versatile processor array for matrix computations has been derived. Finally, some performance characteristics for different variants of implementing this array have been investigated.

References

[1] S. Y. KUNG, *"VLSI Array Processors,"* Prentice-Hall, Englewood Cliffs, N.J., 1988.

[2] J. H. MORENO, and T. Lang, "Matrix Computations on Systolic-Type Meshes," *Computer*, Vol. 23, No. 4, pp.32-51, 1990.

[3] J. RICE, *"Matrix Computations and Mathematical Software,"* McGraw-Hill Book Comp., New York, 1981.

[4] J. G. NASH, and S. HANSEN, "Modified Faddeev Algorithm for Concurrent Execution of Linear Algebraic Operations," *IEEE Trans. Comput.*, Vol. C-37, pp.129-137, 1988.

[5] D. K. FADDEEV, and V. N. FADDEVA, *"Computational Methods of Linear Algebra,"* W. H. Freeman and Company, 1963.

[6] H. Y. H. CHUANG, and G. HE, "A Versatile Systolic Array for Matrix Computations," *Int. Symp. Comput. Archit.*, pp.315-322, 1985.

[7] H. V. D. LE, and M. A. PERKOWSKI, "A New General Purpose Systolic Architecture for Matrix Computations," *Proc. Int. Conf. on Computing and Information, ICCI'89*, Ontario, 1989.

[8] H. V. D. LE, and M. A. PERKOWSKI, "Realization of Extensions to Faddeev algorithm on Array of SIMD Processors', *Proc. IEEE Int. Symp. Circuits and Syst., New Orleans*, pp.2312-2315, May 1990.

[9] M. A. BAYOUMI, P. RAO, and B. ALHALABI, "VLSI Parallel Architectures for Kalman Filtering: An Algorithm Specific Approach," *J. VLSI Signal Process.*, Vol. 4, pp. 147-163, 1992.

[10] M. GENTLEMAN, and H. T. KUNG, "Matrix Triangularisation by Systolic Arrays," *Proc. SPIE*, Vol. 298, pp. 19-26, 1982.

[11] D. SORENSON, "Analysis of Pairwise Pivoting in Gaussian Elimination," *IEEE Trans. Comput*, Vol. C-34, pp. 274-278, 1985.

[12] M. COSNARD, "Designing Parallel Algorithms for Linearly Connected Processors and Systolic Arrays," *Advances in Parallel Computing*, Vol. 1, pp. 273-317, 1990.

[13] R. WYRZYKOWSKI, "Processor Arrays for Matrix Triangularisation with Partial Pivoting," *IEE Proc. E, Comput. Digit. Tech.*, Vol. 139, No.2, pp. 165-169, 1992.

[14] D. I. MOLDOVAN, and J. A. B. FORTES, "Partitioning and Mapping Algorithms into Fixed-Size Systolic Arrays," *IEEE Trans. Comput.*, Vol. C-35, pp. 1-12, 1986.

[15] V. V. VOEVODIN, *"Mathematical Models and Methods in Parallel Computing,"* Nauka, Moscow, 1986 (in Russian).