Joint Conference of 1996 International Computer Symposium
December 19~21, Kaohsiung, Taiwan, R.O.C.

Systematic Array Processors Design for Fraction-free Algorithm

Shietung Peng* , Igor Sedukhin+ and Stanislav Sedukhin*

*) The University of Aizu, Fukushima 965-80, Japan
e-mail: {s-peng, sedukhin}@u-aizu.ac.jp
+)R&D Group, Hiwada Electronic Corp. Fukushima 969-13, Japan

Abstract

The design of systolic array processors for solving linear systems of equations using fraction-free Gaussian elimination method is presented. The design is based on a formal approach which constructs a family of planar array processors systematically. These array processors are synthesized and analyzed. It is shown that some array processors are optimal in the framework of linear allocation of computations and in terms of number of processing elements, number of input/output ports, time of processing, and data pipelining period.

1 Introduction

Integer-preserving transformations for the exact solution of systems of linear equations are studied in this paper. Because integer operations are much more faster than the operations for real numbers, integer-preserving transformation is an effective technique in the areas like symbolic computation and the application areas where input values can be presented as integers and the results are required as rations of integers [3]. Some mathematical software packages (like MATHEMATICA and MAPLE) use techniques of integer-preserving transformations simply because of its efficiency.

In the era of high-performance computing, the design of parallel algorithms and/or application-specific architectures which show high regularity in computations and communication is desired. Systolic/wavefront array processors design is one of candidates with great potential [5]. Traditionally, systolic/wavefront arrays were emphasized for its simplicity for VLSI implementation. However, in the recent study, it was shown that the design methodologies of systolic arrays can be applied elegantly to other models of parallel computations [4].

In this paper, an approach [8] of systematic design/synthesis of systolic arrays which use the fraction-free Gaussian elimination method is demonstrated. The systematic approach makes the algorithmic/architectural design of systolic arrays more attractive.

The rest of the paper is organized as follows. Section 2 gives some theoretical background for the fraction-free Gaussian elimination method [1], and the first parallel algorithm (initial algorithm). In Section 3, the three-dimensional (3-dim) data dependency graph of the initial algorithm and its analysis are given and some problems in the systolic design using the initial algorithm are indicated. A refined parallel algorithm is elaborated in Section 4. In Section 5, systematic design of planar systolic array processors using the refined algorithm are presented and two optimal systolic arrays are demonstrated and analyzed in details. Finally, some concluding remarks and the further research directions are given in the last section.

2 Fraction-free Algorithm

The formal specification of *fraction-free algorithm* is presented below. Let a linear system of equations be given by

$$\mathbf{AX} = \mathbf{B}, \tag{1}$$

where $\mathbf{A} = [a_{ij}]_{n\times n}$, $1 \le i,j \le n$, is a nonsingular matrix with determinant $|A|$, $\mathbf{X} = [x_{ij}]_{n\times(m-n)}$, $1 \le i \le n$, $1 \le j \le m-n$, is a set of unknowns and $\mathbf{B} = [b_{ij}]_{n\times(m-n)}$, $1 \le i \le n$, $n+1 \le j \le m$, is an arbitrary right-hand side matrix. The reduction of extended $[n\times m]$-matrix $\mathbf{A}^{(0)} = \mathbf{A} \oplus \mathbf{B}$ ($\mathbf{A}$ augmented by $\mathbf{B}$) to diagonal form can be done by the recurrence formulas:

$$a_{00}^{(0)} = 1, a_{ij}^{(0)} = \begin{cases} a_{ij}, & 1 \le i \le n,\ 1 \le j \le n; \\ b_{ij}, & 1 \le i \le n, n+1 \le j \le m; \end{cases}$$

$$k = 1,2,...,n,\ 1 \le i \le n,\ k+1 \le j \le m:$$

$$a_{ij}^{(k)} = \begin{cases} a_{kj}^{(k-1)}, & \text{if } i = k; \\ \frac{1}{a_{k-1,k-1}^{(k-1)}} \begin{vmatrix} a_{kk}^{(k-1)} & a_{kj}^{(k-1)} \\ a_{ik}^{(k-1)} & a_{ij}^{(k-1)} \end{vmatrix}, & \text{otherwise.} \end{cases}$$

$$a_{ii}^{(k)} = a_{kk}^{(k)}, 1 \le i \le k-1; \tag{2}$$

Notice that from (2) follows

$$\begin{aligned} a_{ii}^{(k)} &= \frac{1}{a_{k-1,k-1}^{(k-1)}} \begin{vmatrix} a_{kk}^{(k-1)} & a_{ki}^{(k-1)} \\ a_{ik}^{(k-1)} & a_{ii}^{(k-1)} \end{vmatrix} \\ &= \frac{a_{kk}^{(k-1)} \cdot a_{ii}^{(k-1)} - 0}{a_{k-1,k-1}^{(k-1)}} = a_{kk}^{(k-1)} = a_{kk}^{(k)}, \end{aligned}$$

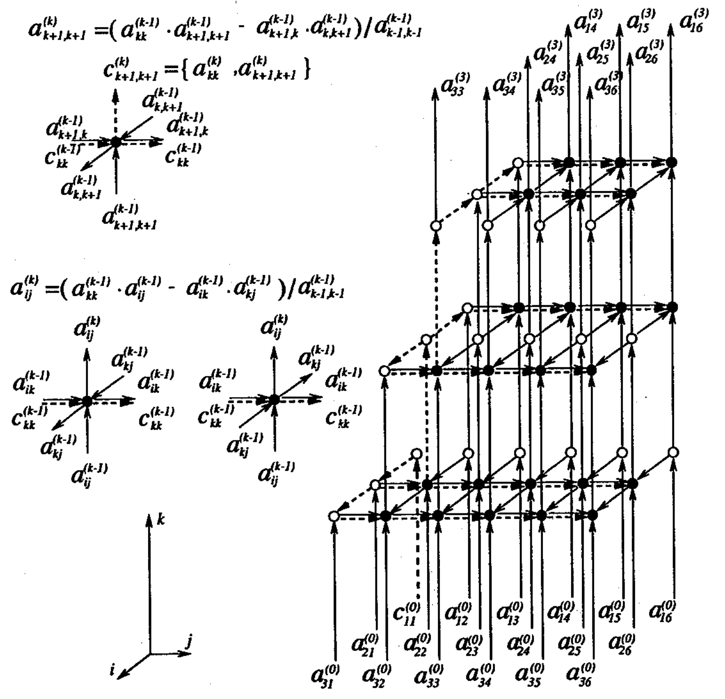
$a^{(k)}_{k+1,k+1} = (a^{(k-1)}_{kk} \cdot a^{(k-1)}_{k+1,k+1} - a^{(k-1)}_{k+1,k} \cdot a^{(k-1)}_{k,k+1}) / a^{(k-1)}_{k-1,k-1}$
$c^{(k)}_{k+1,k+1} = \{a^{(k)}_{kk}, a^{(k)}_{k+1,k+1}\}$
$a^{(k)}_{ij} = (a^{(k-1)}_{kk} \cdot a^{(k-1)}_{ij} - a^{(k-1)}_{ik} \cdot a^{(k-1)}_{kj}) / a^{(k-1)}_{k-1,k-1}$
k
j
i

Figure 1: The data dependence graph of the initial algorithm.

because $a_{ki}^{(k-1)} = 0$ for $i < k$ and $a_{ii}^{(k-1)} = a_{k-1,k-1}^{(k-1)}$ for $k \geq 2$. Note the important property that when all initial $a_{ij}^{(0)}$ are integers, $a_{ij}^{(k)}$ is also an integer, since it is well known that

$$a_{ij}^{(k)} = \begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1k} & a_{1j} \\ a_{21} & a_{22} & \cdots & a_{2k} & a_{2j} \\ \vdots & \vdots & & \vdots & \vdots \\ a_{k1} & a_{k2} & \cdots & a_{kk} & a_{kj} \\ a_{i1} & a_{i2} & \cdots & a_{ik} & a_{ij} \end{vmatrix}, \qquad k < i, j \leq n.$$

The description above is a formal specification of the one-step integer-preserving Gaussian elimination algorithm [1].

After $k = n$ iterations we will have the form

$$\mathbf{A}^{(n)} = \begin{bmatrix} a_{nn}^{(n)} & 0 & \dots & 0 & a_{1,n+1}^{(n)} & \dots & a_{1m}^{(n)} \\ 0 & a_{nn}^{(n)} & \dots & 0 & a_{2,n+1}^{(n)} & \dots & a_{2m}^{(n)} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & a_{nn}^{(n)} & a_{n,n+1}^{(n)} & \dots & a_{nm}^{(n)} \end{bmatrix}$$

where $a_{nn}^{(n)} = a_{nn}^{(n-1)} = |A|$. Then the solution to (1) is given by

$$x_{ij} = a_{i,n+j}^{(n+1)} = a_{i,n+j}^{(n)} / a_{nn}^{(n)},\ 1 \leq i \leq n,\ 1 \leq j \leq m-n.$$

An initial algorithm for solving equation (1) can be derived directly from (2) using indexed, single-assignment format as follows:

```
\ input computations (initialization)
$a_{00}^{(0)} \leftarrow 1$;
for all $1 \leq i \leq n$ do
    $a_{ij}^{(0)} \leftarrow \begin{cases} a_{ij}, & 1 \leq j \leq n; \\ b_{ij}, & n+1 \leq j \leq m; \end{cases}$
\ internal computations
for $k = 1$ to $n$ do
  begin
    $a_{kk}^{(k)} \leftarrow a_{kk}^{(k-1)}$;                                   (3)
    for all $1 \leq i < n, k+1 \leq j \leq m$ do
      $a_{ij}^{(k)} \leftarrow$ if $i = k$ then $a_{kj}^{(k-1)}$ else
        $(a_{kk}^{(k)} \cdot a_{ij}^{(k-1)} - a_{ik}^{(k-1)} \cdot a_{kj}^{(k-1)}) / a_{k-1,k-1}^{(k-1)}$;
  end
\ output computations
for all $1 \leq i \leq n, 1 \leq j \leq m-n$ do
    $x_{ij} = a_{i,n+j}^{(n+1)} = a_{i,n+j}^{(n)} / a_{nn}^{(n)}$;
```

From (3) it is easy to see that the internal computations of fraction-free algorithm requires

$$N = \sum_{k=1}^{n} n(m-k) = mn^2 - n^3/2 + O(mn)$$

steps, where each step contains at most 2 multiplications, one substraction and one integer division. The output computations is not integer-preserving and require $n(m-n)$ divisions. Thus this algorithm is suitable for solving the linear systems with integer coef-

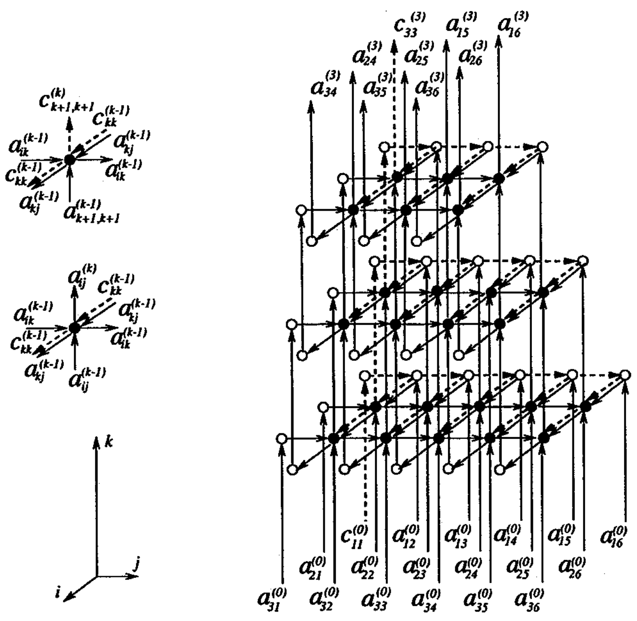

Figure 2: The data dependence graph of the regular algorithm.

ficients since the algorithm needs only integer operations which can be computed much faster than operations with real numbers needed by other algorithms in which fractions are generated as intermediate results.

3 Dependence Analysis of the Initial Algorithm

As it directly follows from (3), the *index* or *iteration space* $\mathcal{I} = \mathbf{Z}^3$ of the initial algorithm consists of the index subsets of input, internal and output computations, i.e.

$$\mathcal{P} = \mathcal{P}_{in} \cup \mathcal{P}_{int} \cup \mathcal{P}_{out},$$

where

$$\begin{aligned}
\mathcal{P}_{in} &= \{(i,j,0)^T \cup (0,0,0)^T \mid 1 \le i \le n, \\
&\quad 1 \le j \le m\} \subseteq \mathbf{Z}^2; \\
\mathcal{P}_{int} &= \{(i,j,k)^T \mid 1 \le k \le n,\ 1 \le i \le n, \\
&\quad k+1 \le j \le m\} \subseteq \mathbf{Z}^3; \\
\mathcal{P}_{out} &= \{(i,n+j,n+1)^T \mid 1 \le i \le n, \\
&\quad 1 \le j \le m-n\} \subseteq \mathbf{Z}^2
\end{aligned}$$

where $\mathbf{Z}$ is a set of integer numbers. Each index point $p = (i,j,k)^T \in \mathcal{P}$ is associated with a single input/internal or output computation.

A *data dependence vector* Θ is the difference between the index points where a variable is used as *input variable* and the index point where that variable was generated as *output variable*. From (3) it follows that for each output variable $a^{(k)}_{ij}$, i.e. for any index point $p = (i,j,k)^T \in \mathcal{P}_{int}$, the data dependence vectors of the input variables $a^{(k-1)}_{kj}$, $a^{(k-1)}_{kk}$, $a^{(k-1)}_{ij}$, $a^{(k-1)}_{ik}$, $a^{(k-1)}_{k-1,k-1}$ are $\Theta_{kj} = (i-k,0,1)^T$, $\Theta_{kk} = (i-k,j-k,1)^T$, $\Theta_{ij} = (0,0,1)^T$, $\Theta_{ik} = (0,j-k,1)^T$, $\Theta_{k-1,k-1} = (i-k+1,j-k+1,1)^T$, respectively. It could be seen, that only Θ_{ij} is a constant dependence vector and others reveal the *global data dependence.* For example, since $\Theta_{kk} = (i-k,j-k,1)^T$, each output variable $a^{(k)}_{ij}, 1 \le i \le n,\ k+1 < j \le m,\ i \ne k$, depends upon the input variable $a^{(k-1)}_{kk}$, i.e. all of $(n-1)(m-k)$ index points, that lie on k-th $\left\langle \vec{i}, \vec{j} \right\rangle$-plane, require the same instance of variable $a^{(k-1)}_{kk}$ from $(k-1)$-th plane. Because $1 \le i, k \le n$, the vectors Θ_{kk}, $\Theta_{k-1,k-1}$ and Θ_{kj} may have positive as well as negative values of the indexes. Hence, the corresponding variables will be transmitted in opposite directions. Since the variables $a^{(k-1)}_{k-1,k-1}$ and $a^{(k-1)}_{kk}$ should be translated to all internal computations of kth iteration it is convenient to combine two variables into one which will be formed in index point $(k+1,k+1,k)^T$, i.e. $c^{(k)}_{k+1,k+1} = \{a^{(k)}_{kk}, a^{(k)}_{k+1,k+1}\}$ with initial value of $c^{(0)}_{11} = \{a^{(0)}_{00} = 1, a^{(0)}_{11}\}$.

For this algorithm, a non-local *data dependence graph* (DG) can be constructed in 3-dim index space $\left\langle \vec{i}, \vec{j}, \vec{k} \right\rangle$, representing the locations of the computations in the space (as the index points) and the data dependencies (as the arcs) between the computations of the algorithm. This non-local DG can then be con-

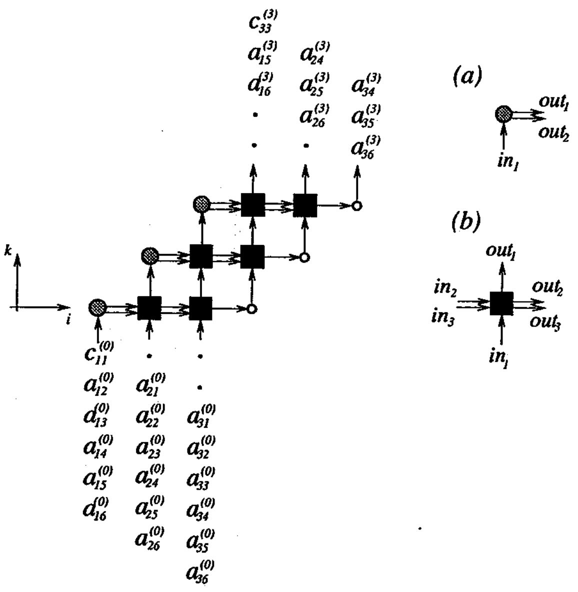

Figure 3: The array processor $S_{(0,0,1)}$.

verted to a local DG by localization via pipelining all global data dependencies.

The localized DG Γ_1 of the algorithm (3) is shown in Fig. 1 for the case of $n = 3$ and $m = 6$ (without the output computations). The *longest path* in this DG is any longest path between the index points $p_{\min}$ and $p_{\max}$, where $p_{\min} = (1,1,1)^T$ and $p_{\max} = (1,m,n)^T$. It can be shown that the *length* $\mathcal{L}$ of the longest path is $\mathcal{L}(p_{\min}, p_{\max}) = 4(n-1) + (m-n) = m + 3n - 4$.

Since $p_{\min}^{(k)} = (k,k,k)$ we have $\mathcal{L}(p_{\min}^{(k-1)}, p_{\min}^{(k)}) = 3$. From this, a *timing* (*step*) *function* $\mathbf{step}(p) : \mathcal{P}_{int} \rightarrow \mathbf{Z}_+$ which assigns a computational time step to each index point $p \in \mathcal{P}_{int}$ can be defined as follows:

$$\mathbf{step}(p) = |i-k| + |j-k| + 3k - 3,$$

bearing in mind that $\mathbf{step}(p_{\min}) = 0$, and assuming that the computation and the corresponding local data communication are realized in each index point in one unit time-step. The internal computation set of the algorithm can be evaluated in the minimal possible time

$$T_{\min}(\Gamma_1) = \mathbf{step}(p_{\max}) = m + 3n - 4,$$

which equals to the length of the longest path $\mathcal{L}(p_{\min}, p_{\max})$ in the DG of the initial algorithm.

In order to find an optimal design of a systolic array processor from the large space of the admissible solutions, the optimality criteria shall include many factors. Some typical factors are ([6],[7])

- *Computation Time* (T);
- *Array Size* (P);
- *Block Pipelining Period* (β) : the time interval between the initiations of two successive problem instances by the processor array;
- *Data Pipelining Period* (α) : the time interval separating the neighboring items of input or output data flow;
- *I/O Data Flow Formats* , i.e. the structure in which flows of the input/output data of the problem are organized;
- *Number of I/O Ports.*

Note that for $1 \le i,k \le n$, $k \le j \le m$, we have

$$\begin{aligned} \Delta t &= \mathbf{step}(i,j,k+1) - \mathbf{step}(i,j,k) \\ &= \begin{cases} 1, & \text{if } (i > k) \text{ and } (j > k); \\ 3, & \text{if } (i \le k) \text{ and } (j > k); \end{cases} \end{aligned} \quad (4)$$

Hence, for the time consistency of processing and communications, it is necessary to add $(\Delta t - 1)$ unit delays (FIFO buffers) to each arc that connects two neighboring nodes of the 3-dim DG laying along the $\vec{k}$ axis.

Having the definition of the timing function $\mathbf{step}(p)$ the notion of a *flow velocity vector* of a variable v along a direction $\vec{e}_v$ can be introduced [4]:

$$\mathbf{flow}(v) = \frac{q-p}{\mathbf{step}(q) - \mathbf{step}(p)},$$

where $p, q \in P_{int}$ such that the variable v is used firstly at the index point p and then at the point q, i.e. $\mathbf{step}(p) < \mathbf{step}(q)$.

As we assume $\mathbf{step}(p_{\min}) = 0$, an extension of the domains of input/output computations is necessary to obtain correct allocation of the input/output data flows on the processing space. It can be done as follows.

- For the input data:

  $$p_{in} = p_0 - (\mathbf{step}(p_0) + 1) \cdot \mathbf{flow}(v),$$

  where $p_0 \in \mathcal{P}_{in}, v$ is the input variable of the algorithm corresponding to the input computation at the index point $p_0, \mathbf{step}(p_{in}) = -1$;

- For the output data:

  $$q_{out} = q_0 + (\mathbf{step}(p_{\max}) - \mathbf{step}(q_0) + 1) \cdot \mathbf{flow}(u),$$

  where $q_0 \in \mathcal{P}_{out}, u$ is the output variable of the algorithm corresponding to the output computation at the index point $q_0, p_{\max} \in \mathcal{P}_{int}, \mathbf{step}(q_{out}) = \mathbf{step}(p_{\max}) + 1$.

After we apply these transformations to the domain of input/output computations we will obtain the extended data dependence graph (EDG) of the algorithm. This 3-dim EDG is ready to be mapped onto 2-dim space.

The spatial allocation of the set of computations given in the space $\mathcal{I} = \mathbf{Z}^3$ onto the space $\mathcal{S} = \mathbf{Z}^2$ is accomplished by an allocation function

$$\mathbf{place}(p) : \mathcal{I} \to \mathcal{S}.$$

This function is given for each computation-node $p \in \mathcal{P}$ from EDG

- either (when $p \in \mathcal{P}_{int}$) $\mathcal{S}$-coordinates of a PE, which will execute the computation and communication of the index point p at the $\mathbf{step}(p)$;
- or (when $p \in \mathcal{P}_{in} \cup \mathcal{P}_{out}$) coordinates of the index point where the element p of the input/output data will be allocated through the recurrent steps.

A linear form of the allocation function is used:

$$\mathbf{place}(p) = \mathbf{\Lambda}_\eta \cdot p,$$

where $\mathbf{\Lambda}_\eta$ is a (2×3)-matrix of the linear transformation corresponding to a *projection direction* $\eta \in$ ker $\mathbf{\Lambda}_\eta$ and which has the **rank** $\mathbf{\Lambda}_\eta = 2$.

Along all possible projection directions which the cubic EDG may be projected onto the plane, only those are *admissible* that

1) keep local data dependencies between computations;
2) do not map any index points $p, q \in \mathcal{P}_{int}$, such that $\mathbf{step}(p) = \mathbf{step}(q)$, onto the same PE.

The problems are caused by the irregularity of the algorithm and the DG it generated. The major problems concerning the initial algorithm and it's DG are:

- some data are transmitted along opposite edges in $\vec{i}$ direction, thus prevent projections in this direction;
- the arc-delays along $\vec{k}$-axis vary as shown by (4), thus complicate the control of computation/communication.

To design optimal array processors, we present a new algorithm which eliminates the problems mentioned above using an innovative reindexing technique in the next section.

4 The Design of the Regular Algorithm

One way to resolve the irregularity problem of the initial algorithm (3) is to reindex the nodes in the DG. The reindexing is as follows: at the k-th iteration, we shift the k-th row, $(a_{k,k+1}^{(k-1)}, ..., a_{km}^{(k-1)})$, to $(n+k)$-th row. In this pattern the propagating variables move only in the positive directions. Based on the above reindexation scheme we can rewrite the algorithm (3) in the following form (*affine recurrent equations* scripted as nested loops):

```
\ input computations (initialization)
$a_{00}^{(0)} \leftarrow 1;$
for all $1 \leq i \leq n$ do
    $a_{ij}^{(0)} \leftarrow \begin{cases} a_{ij}, & 1 \leq j \leq n; \\ b_{ij}, & n+1 \leq j \leq m; \end{cases}$
\ internal computations
for $k = 1$ to $n$ do
    begin
        $a_{kk}^{(k)} \leftarrow a_{kk}^{(k-1)};$                                        (5)
    for all $k < i \leq n+k, k < j \leq m$ do
        $a_{ij}^{(k)} \leftarrow$ if $i = n+k$ then $a_{kj}^{(k-1)}$ else
        $(a_{kk}^{(k)} \cdot a_{ij}^{(k-1)} - a_{ik}^{(k-1)} \cdot a_{kj}^{(k-1)})/a_{k-1,k-1}^{(k-1)};$
    end
\ output computations
for all $1 \leq i \leq n, 1 \leq j \leq m-n$ do
    $x_{ij} = a_{i+n,j+n}^{(n+1)} = a_{i+n,j+n}^{(n)}/a_{nn}^{(n)};$
```

Notice that after $k = n$ iteration we obtain the form

$$\mathbf{A}^{(n)} = \begin{bmatrix} 0 & . & 0 & a_{11}^{(1)} & a_{1,n+1}^{(n)} & . & a_{1,m}^{(n)} \\ 0 & . & 0 & a_{22}^{(2)} & a_{2,n+1}^{(n)} & . & a_{2,m}^{(n)} \\ . & . & . & . & . & . & . \\ 0 & . & 0 & a_{nn}^{(n)} & a_{n,n+1}^{(n)} & . & a_{n,m}^{(n)} \end{bmatrix}, \quad (6)$$

which is characterized by the desirable property of locality for output computations. Note that the elements $a_{kk}^{(k)}$ in (6) are now the leading principal minors of $\mathbf{A}$.

The index subspaces of the algorithm (5) are described as follows:

$$\begin{aligned} \mathcal{P}_{in} &= \{(i,j,0)^T |\ 1 \leq i \leq n,\ 1 \leq j \leq m\} \subseteq \mathbf{Z}^2; \\ \mathcal{P}_{int} &= \{(i,j,k)^T |\ 1 \leq k \leq n,\ k < i \leq n+k, \\ &\quad k < j \leq m\} \subseteq \mathbf{Z}^3; \\ \mathcal{P}_{out} &= \{(i+n, j+n, n+1)^T |\ 1 \leq i \leq n, \\ &\quad 1 \leq j \leq m-n\} \subseteq \mathbf{Z}^2. \end{aligned}$$

The data dependence vectors of the input variables $a_{kj}^{(k-1)}$, $a_{kk}^{(k-1)}$, $a_{ij}^{(k-1)}$, $a_{ik}^{(k-1)}$, $a_{k-1,k-1}^{(k-1)}$ are the same as for the irregular algorithm (3), but since now $i > k$, the vectors $\Theta_{kj} = (i-k, 0, 1)^T$, $\Theta_{kk} = (i-k, j-k, 1)^T$ and $\Theta_{k-1,k-1} = (i-k+1, j-k+1, 1)^T$ may only have positive values of the indexes. Hence, the corresponding variables will be transmitted in one direction.

The dependencies localization procedure [8] applied to the initial affine recurrent equations (5) results in the *uniform recurrent equations* which have the following form:

$$
\begin{array}{l}
/ \text{ input computations} \\
\left\{ \begin{array}{l} (i=1, j=1) : y(p) \leftarrow \{1, a_{11}^{(0)}\}; \\ (i \neq 1 \text{ or } j \neq 1) : y(p) \leftarrow a_{ij}^{(0)}; \end{array} \right. \quad p \in \mathcal{P}_{in} \\
/ \text{ internal computations} \\
\left\{ \begin{array}{l}
(k < i < k+n, k < j < m) : \\
\quad : \left\{ \begin{array}{l} (i = k+1) : \\ \quad : x_1(p) \leftarrow y(p - e_1); \\ (i \neq k+1) : \\ \quad : x_1(p) \leftarrow x_1(p - e_1); \\ x_2(p) \leftarrow x_2(p - e_2); \\ x_3(p) \leftarrow y(p - e_3); \\ y(p) \leftarrow \frac{x_1^{12}(p) \cdot x_3(p) - x_2(p) \cdot x_1^2(p)}{x_1^{11}(p)}; \end{array} \right. \\
(k \leq i < k+n, j = k) : \\
\quad : \left\{ \begin{array}{l} x_3(p) \leftarrow y(p - e_3); \\ y(p) \leftarrow x_3(p); \end{array} \right. \qquad p \in \mathcal{P}_{int} \\
(i = k+n, k < j < m) : \\
\quad : \left\{ \begin{array}{l} x_1(p) \leftarrow x_1(p - e_1); \\ y(p) \leftarrow x_1^2(p); \end{array} \right. \\
(i = k, k < j < m) : \\
\quad : \left\{ \begin{array}{l} (j = k+1) : \\ \quad : x_2(p) \leftarrow y(p - e_2); \\ (j \neq k+1) : \\ \quad : x_2(p) \leftarrow x_2(p - e_2); \\ x_3(p) \leftarrow y(p - e_3); \\ y(p) \leftarrow \{x_2(p), x_3(p)\}; \end{array} \right.
\end{array} \right. \\
/ \text{ output computations} \\
\left\{ a_{ij}^{(n)} \leftarrow y(p - e_3); \ p \in P_{out} \right.
\end{array}
\tag{7}
$$

where $e_1 = (1,0,0)^T, e_2 = (0,1,0)^T, e_3 = (0,0,1)^T$ are local data dependence vectors, $y(p)$ is an *output* variable and $x_1(p), x_2(p), x_3(p)$ are *input* variables of the computation at the point $p \in \mathcal{P}$. Note that in uniform recurrent equations we use set-variables of the forms $v = \{v^1, v^2\}$, $w = \{\{w^{11}, w^{12}\}, w^2\}$, and an operation to join two variables in a set (see $i = k, k < j < m$ case).

The localized regular DG Γ_2 corresponding to the equations (7) is depicted on Fig. 2 for the case of $n = 3$, $m = 6$. In DG Γ_2, variables $a_{kk}^{(k-1)}$ and $a_{k-1,k-1}^{(k-1)}$ are denoted together as $c_{kk}^{(k-1)}$ since they are distributed along the same route. As above, for the distribution of global variables $a_{kk}^{(k-1)}$ and $a_{k-1,k-1}^{(k-1)}$ on each k-th plane we have some degree of freedom for the selection of the localization contour from all possible patterns. This selection may be done according to the chosen projection direction of the DG. One of the desirable localization contours (not the best one for some projections) is shown in Fig. 1 and Fig. 2 by dashed lines. The DG of the regular algorithm without output computations contains $n(n-1)(2m-n-1)/2$ nodes associated with the internal computations (fraction-free operations) and some number of time-delays nodes associated with the redefinition of global (transmitted) variables. Notice that this redefinition will expand a bit the subset of internal computations $\mathcal{P}_{int}$.

For the regular DG the *timing function* $\mathbf{step}(p)$ can be specified in the linear form as $\mathbf{step}(p) = \lambda \cdot p + \gamma$. This function defines a set of hyperplanes orthogonal to the *schedule vector* λ on the index space of the algorithm. It is easy to show that the timing function of the *minimal form* for the regular DG is

$$\mathbf{step}(p) = i + j + k - 3$$

for any $p \in \mathcal{P}_{int}$, taking into account that $\mathbf{step}(p_{\min}) = 0$, where $p_{\min} = (1,1,1)^T \in \mathcal{P}_{int}$. Thus, the schedule vector λ is $(1,1,1)$ and γ is -3. Because

$$\Delta t = \mathbf{step}(i,j,k) - \mathbf{step}(i,j,k-1) = 1,$$

no additional delays in between k-th and $(k-1)$-th layers are needed.

It can be proved that the *length of the longest path* in the regular DG is

$$\mathcal{L}(p_{\min}, p_{\max}) = 3(n-1) + m,$$

where $p_{\max} = (2n, m, n) \in \mathcal{P}_{int}$. The set of all computations can be evaluated in the minimal time

$$T_{\min}(\Gamma_2) = \mathbf{step}(p_{\max}) = 3(n-1) + m,$$

which equals to the length of the longest path in the DG of the regular algorithm.

5 Optimal Array Processors

As mention before, the *timing* and *allocation functions* must guarantee that each PE executes at most one computation, which associated with any index point of DG, at any given time step. For the regular DG with a linear timing function this requires the condition

$$\lambda \cdot \eta \neq 0 \tag{8}$$

to be held.

According to [8] for the 3-dim DG of the regular algorithm there are 17 possible *projection directions* that keep the local communications and define systolic solutions. However, only 13 directions are admissible for the DG Γ_2 according to the condition (8). The system of uniform recurrent equations (7) are used for CAD tool [8] which automatically generates all admissible projects of array processors for the given algorithm and presents them graphically with necessary space-time analysis provided. With the help of this graphic analysis tool, characteristics of each project can be investigated and generalized. The number of PEs, delays, I/O ports and the data pipelining period $\alpha = |\lambda \cdot \eta|$ of all admissible projects are listed in the following Table.

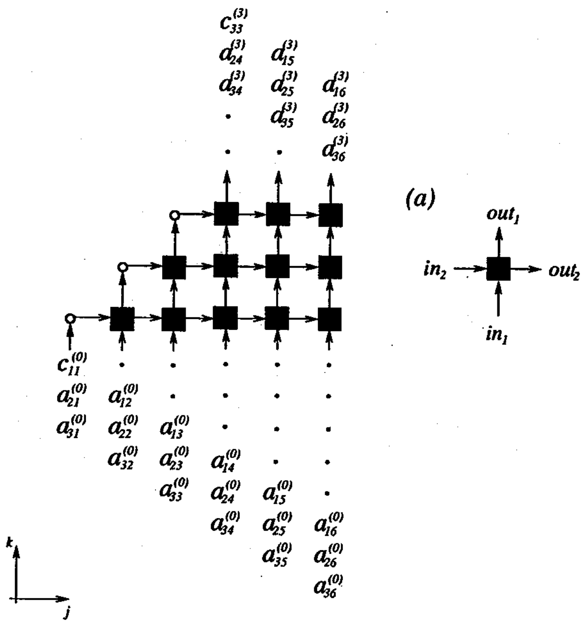
$c_{33}^{(3)}$
$d_{24}^{(3)}$ $d_{15}^{(3)}$
$d_{34}^{(3)}$ $d_{25}^{(3)}$ $d_{16}^{(3)}$
$d_{35}^{(3)}$ $d_{26}^{(3)}$
$d_{36}^{(3)}$
(a)
out_1
in_2
out_2
in_1
$c_{11}^{(0)}$
$a_{21}^{(0)}$ $a_{12}^{(0)}$
$a_{31}^{(0)}$ $a_{22}^{(0)}$ $a_{13}^{(0)}$
$a_{32}^{(0)}$ $a_{23}^{(0)}$ $a_{14}^{(0)}$
$a_{33}^{(0)}$ $a_{24}^{(0)}$ $a_{15}^{(0)}$
$a_{34}^{(0)}$ $a_{25}^{(0)}$ $a_{16}^{(0)}$
$a_{35}^{(0)}$ $a_{26}^{(0)}$
$a_{36}^{(0)}$
k
j

Figure 4: The array processor $S_{(1,0,0)}$.

From this Table, it can be shown that the only solutions, generated by mapping the DG Γ_2 along the projection directions $\eta_1 = (0,1,0)^T$ and $\eta_2 = (1,0,0)^T$, i.e. along $\vec{j}$-axis and $\vec{i}$-axis, are characterized in aggregate by the minimal numbers of PE's and I/O ports, and the number of delays. The array processor $S_{(0,1,0)}$ is generated by mapping the EDG Γ_2 along $\vec{j}$ direction, i.e. along projection direction $\eta_1 = (0,1,0)^T$. This 2-dim rhombic array processor $S_{(0,1,0)}$ (see Fig. 3 for the case of $n = 3$, $m = 6$) consists of

$$P_{(0,1,0)} = n(n-1)$$

orthogonally connected PEs, n controlled delays (CDs) and n ordinary delays (OD). Thus a size of array is independent of m. The initial matrix $\mathbf{A}^{(0)} = [a_{ij}^{(0)}]_{n\times m}$ is loaded into the array processor column by column.

The functions of each $\mathrm{PE}_{i,k}$ and $\mathrm{CD}_{k,k}$ can be presented as follows (variable r represents internal register in each PE and CD):

```
CD_{k,k} (Fig. 3(a))
  $1 \le k \le n$:
/ first step
$r \leftarrow in_1$;
/ next steps
$\{out_2 \leftarrow r; out_1 \leftarrow in_1\}$
```

```
PE_{i,k}, (Fig. 3(b))
$1 \le k \le n,\ k < i < n+k$:
/ first step
$r \leftarrow in_1$;
/ next steps
$\{out_1 \leftarrow \frac{in_3^1 \cdot in_1 - r \cdot in_2}{in_3^2}$;
$out_2 \leftarrow in_2; out_3 \leftarrow in_3\}$
```

The rhombic array processor $S_{(0,1,0)}$ simulates the 3-dim DG Γ_2 without *time extension*, i.e. it solves the problem in time $T_{(0,1,0)}(1,n,m) = T_{\min}(\Gamma_2) = 3(n-1)+m$. The *data pipelining period* is $\alpha = |\lambda \cdot \eta_1| = 1$ and the *block pipelining period* is $\beta = m+1$, i.e. the next task can be pushed into this array processor after $m+1$ time steps. The number of I/O ports is $2n$. Also it is evident that for l tasks the time of processing is $T_{(0,1,0)}(l,n,m) = 3(n-1) + l(m+1) - 1$.

Another optimal design can be obtained by mapping the EDG along $\vec{i}$ axis, i.e. along the projection direction $\eta_2 = (1,0,0)^T$. The corresponding systolic array processor and input/output data flows are shown in Fig. 4 for the case of $n = 3$ and $m = 6$. The array uses

$$P_{(1,0,0)} = n(2m-n-1)/2$$

processing elements $\mathrm{PE}_{j,k}$ (see Fig. 4 (a)), $1 \le k \le n+1, k+1 \le j \le m$, and n ordinary delays. It is not difficult to show that array processor $S_{(1,0,0)}$ has smaller number of PEs than $S_{(0,1,0)}$ and all other admissible projects if

$$n < m < (3n-1)/2 \approx 1.5n.$$

The functions of each $\mathrm{PE}_{j,k}$ can be presented as follows (r_1, r_2, r_3 are the internal registers in each PE):

```
/ first step
$r_1 \leftarrow in_1; r_2 \leftarrow in_2^1; r_3 \leftarrow in_2^2$;
/ next steps
  $\{out_1 \leftarrow (r_2 \cdot in_1 - in_2 \cdot r_1)/r_3$;
    $out_1 \leftarrow in_1; out_2 \leftarrow in_2\}$
```

The number of I/O ports in this design is $2m - n$. The initial expanded matrix $\mathbf{A}_0 = [a_{ij}^{(0)}]_{n\times m}$ is

Project	PEs	Delays	I/O ports	α
$S_{(0,0,1)}$	$2m(n-1)-n(n+3)/2+2$	$2m+n-2$	$-$	1
$S_{(0,1,0)}$	$n(n-1)$	$2n$	$2n$	1
$S_{(0,1,1)}$	$2m(n-1)-n(n+3)/2+2$	$m+3n-1$	$2n$	2
$S_{(1,0,0)}$	$n(2m-n-1)/2$	n	$2m-n$	1
$S_{(1,0,1)}$	$(n-1)(m-1)$	$2m+n-2$	$2m-n$	2
$S_{(1,1,0)}$	$(m-2)(m-1)/2+n(n-1)$	$2n$	$2m+n-2$	2
$S_{(1,1,1)}$	$(n-1)(m-1)$	$2m+n-2$	$2m+n-2$	3
$S_{(1,-1,1)}$	$(m+n-2)(n-1)$	$2m+3n-4$	$2m+n-2$	1
$S_{(-1,1,1)}$	$(n-1)(3m-n-3)$	$2m+3n-4$	$2m+n-2$	1
$S_{(1,1,-1)}$	$(m-2)(3n-2)$	$2m+3n-4$	$2m+n-2$	1
$S_{(1,1,2)}$	$6mn-n^2-4n-4m$	$4m+2n-2$	$2m+n-2$	4
$S_{(-1,1,2)}$	$8mn-4m-3n^2-3n+5$	$4m+4n-7$	$2m+n-2$	2
$S_{(1,-1,2)}$	$6mn-7n-2n^2-4m+4$	$4m+2n-4$	$2m+n-2$	2

Table 1: Admissible projects

loaded into array row by row. As it is shown in Fig. 3 by dashed lines the global variable $c_{kk}^{(k)} = \{a_{kk}^{(k-1)}, a_{k-1,k-1}^{(k-1)}\}$ is transmitted mainly along the projection direction $\eta_2 = (1,0,0)^T$ and thus, after mapping, it will be stored in the internal registers r_2 and r_3 of the corresponding PE's. The *total computation time* is

$$T_{(1,0,0)}(1,n,m) = T_{\min}(\Gamma_2) = 3(n-1)+m,$$

i.e. the array processor will simulate the DG without time extension. This processor supports a noninterleaved *block pipelining period* of $\beta = n$ and a *data pipelining period* of $\alpha = |\lambda \cdot \eta_2| = 1$, so successive computations can be repeated every n time steps.

6 Conclusion

In this paper the design of systolic array processors for solving systems of linear equations based on a fraction-free algorithm were presented. A new regular iterative algorithm whose dependency graph meets local communication/computation requirements, while remaining input/output equivalent to the initial algorithm is the key of the design. The reindexing technique used in this paper should be applicable to other classes of algorithms for designing efficient, special-purpose array processors. The proposed array processors were derived by the systematic design methodology [8]. It is interesting to investigate the possibility of using non-linear scheduling for reducing number of PEs in optimal array processors generated by linear scheduling [2]. It is also interesting to implement the algorithm on high-performance parallel computers for performance evaluation, error analysis, and empirical study of scalability and benchmark.

References

[1] Bareis E.H. Sylvester Identity and Multistep Integer-Preserving Gaussian Elimination. *Mathematics of Computation*, No. 22, 1968, pp. 565–578.

[2] Clauss Ph., Mongenet C., Perrin G.R. Synthesis of size-optimal toroïdal arrays for the algebraic path problem. *Proc. of the International Workshop Algorithms and Parallel VLSI Architectures II*, France, June 3-6, 1991, Elsevier Publisher, 1992, pp. 199–210.

[3] Fox L. *An Introduction to Numerical Linear Algebra.* Oxford University Press, New York, 1965.

[4] Huang C.H., Lengauer Ch. The derivation of systolic implementations of programs. *Acta Informatica.* No. 24, 1987, pp. 595–632.

[5] Kung S.Y. *VLSI Array Processors.* Prentice Hall, 1988.

[6] Kung S.Y., Lo S.C., Lewis P.S. Optimal systolic design for the transitive closure and the shortest path problems. *IEEE Trans. on Computers.* Vol. C-36, No. 5, 1987, pp. 603–614.

[7] Sedukhin S.G. Design and analysis of systolic algorithms for the algebraic path problem. *Computers and Artificial Intelligence.* Vol. 11, No. 3, 1992, pp. 269–292.

[8] Sedukhin S.G., Sedukhin I.S. Systematic approach and software tool for systolic design. *Lecture Notes in Computer Science*, Number 854, Buchberger B. and Volkert J. eds., Springer-Verlag, 1994, pp. 172–183. (`http://gemini.u-aizu.ac.jp/HPCC/S4CAD/`)