Bull. Nov. Comp. Center, Comp. Science, 1 (1993)

S4CAD: a software tool for synthesis, analysis and modeling of systolic structures

S.G. Sedukhin and I.S. Sedukhin

This paper presents the S4CAD software tool which allows to synthesize and analyse a set of admissible systolic arrays for the given matrix algorithm. A systematic approach to the design is presented as a theoretical background of the S4CAD. The tool runs under graphical operating environment MICROSOFT WINDOWS 3 placing at the user's disposal convenient means for evaluation and choosing an optimal structure observing designer's requirements, e.g., computing time, number of processing elements, structure topology, number of external pins, data flows formats, data pipelining period etc. A number of basic parametrized algorithms of linear algebra and graph theory have been included in the S4CAD library, and as an example the design of systolic structures for the transitive closure algorithm is shown in the paper.

1. Introduction

Investigations in the field of the formal synthesis of parallel VLSI-structures and parallelizing compilers usually require an automated design tool. In particular, when systolic algorithms claiming to be efficiently produced in VLSI or efficiently mapped on a massive-parallel computer architecture are designed, the following demands are made on the automated design tools:

1. ability to produce a data dependence graph given an algorithm and problem size parameters;
2. performing equivalent transformations of the dependence graph, which keep the operational precedence and observe technological or architectural constraints;
3. mapping the dependence graph on admissible processor arrays of the systolic architecture, elements of which belong to the space of given dimension;
4. ability to simulate the processes going on in the processor array during its activity period;

5. availability of analysis means which help to estimate algorithm realizations in the different processor arrays;
6. availability of an interface with lower level packages (i.e., silicon compilers and parallel language compilers).

It is usually desired to have an interactive, friendly user interface with such design tools, especially on the choice optimization stage.

The existing tools aiming at the design of parallel fine-grain structures generally realize isolated stages of the design process. For instance, the following automated design tools may be named: ADVIS [9], DIASTOL [12], SYSTOL [10], SYSTARS [11], PRESAGE [18], VACS [7] – tools that concentrate on the systolic structures synthesis; DECOMPOSER [6] – tool that handles the partitioning and the more advanced software tool for the VLSI-structures design – ARREST [2].

In this paper a software tool for synthesis, analysis and modeling of systolic structures and algorithms – the S4CAD is presented. The paper is organized as follows:

Section 2 briefly presents a formal approach to synthesis and analysis of systolic structures;

Section 3 is an overview of the main capabilities of the S4CAD tool;

Section 4 illustrates the design process of optimal systolic structures on the example of the transitive closure algorithm;

Section 5 is devoted to development perspectives of the S4CAD in the frame of the recent research.

2. Formal approach to the design

Traditionally matrix algorithms are represented by a *system of recurrent equations*. In this representation each variable is denoted by a *name* and by *index variables*, which may only be integers and which uniquely determine the desired variable. The number of index variables defines dimension of an *index space* $\mathcal{I} = \mathsf{Z}^n$ and the admissible range of this variables defines an *internal computations domain* $\mathcal{P}_{int}$, which is a bounded convex polyhedron for finite algorithms. *Initial values* have to be assigned to recurrent variables before equations for any recurrent step can be defined. These initial assignments form an *input computations domain* $\mathcal{P}_{in}$ and, respectively, *final assignments* form an *output computations domain* $\mathcal{P}_{out}$. The set

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

is called *computations domain* of the algorithm.

The original algorithm representation is taken in the form of a system of *affine recurrent equations*[1]

$$y(p_0) = f(y(p_0 - \Theta_1(p_0)), \dots, y(p_0 - \Theta_m(p_0))), \tag{1}$$

where $p_0 \in \mathcal{P}$; f is an unambiguous function strictly depending on its arguments and having the complexity $O(1)$ (in the common case, f may depend upon p_0); $y(p)$ denotes the computation of f function at the point $p \in \mathcal{P}$; $\Theta_i(p_0) \in \mathcal{I}$, $i \in \{1, \dots, m\}$ designates a *direct data dependence vector* (DDD-vector), which defines the dependence of the computation at the point p_0 on the computation at the point $p_i = p_0 - \Theta_i(p_0)$ and which fits the following condition

$$p_i = p_0 - \Theta_i(p_0) = A_i \cdot p_0 + b_i, \tag{2}$$

where A_i is a constant $(n \times n)$-matrix, b_i denotes a constant column-vector of n components.

The $\rho(\vec{x}, \vec{y}) = \max_i |x_i - y_i|$ metrics introduced in the index space $\mathcal{I}$ defines the notion of a neighborhood for the points of the space $\mathcal{I}$.

The most common case for the majority of the algorithms of linear algebra, graph theory, etc. is the case when $n \leq 3$, i.e., $\mathcal{P} \subset \mathcal{I} \subseteq \mathsf{Z}^3 = \{(i,j,k)^\top \mid i,j,k \in \mathsf{Z}\}$, where Z is the set of integers. Usually, the index variable k is used to denote a recurrent step. In this case, for any point $p \in \mathcal{P}$ vector $\Theta_3(p) = [0,0,1]^\top$ is used for *reverse* and $\Theta_3(p) = [0,0,-1]^\top$ for *normal* recurrence. Below the assumption $\mathcal{I} = \mathsf{Z}^3$ is made, the extension to $\mathcal{I} = \mathsf{Z}^n$ is considered trivial.

The equations system (1) with the dependencies (2) may be rewritten taking into account previously made assumption:

$$\begin{cases} x_1(p) \leftarrow y(p - \Theta_1(p)), \\ x_2(p) \leftarrow y(p - \Theta_2(p)), \\ x_3(p) \leftarrow y(p - \Theta_3), \\ y(p) \leftarrow f(x_1(p), x_2(p), x_3(p)), \end{cases} \tag{3}$$

where $y(p)$ is an *output* and $x_1(p), x_2(p), x_3(p)$ are *input* variables of the computation at the point p; $\Theta_1(p), \Theta_2(p), \Theta_3$ are DDD-vectors; f is a function, which is defined at the point p iff all input variables were defined. For unambiguityness $y(p - \Theta_3)$ is considered a *recurrent variable*.

The equations system (3) defines a *direct data dependencies graph* (DDD-graph)

$$\mathcal{G} = (\mathcal{P}_{int}, \{\Theta_1(p), \Theta_2(p), \Theta_3\}_{p \in \mathcal{P}_{int}}).$$

[1]Or in the equivalent form of a nested loop program

For further analysis graph $\mathcal{G}$ is supplemented by the input-output nodes with the necessary arcs.

The equations system (3) (and as a more illustrative form DDD-graph $\mathcal{G}$) allows revealing an important peculiarity of the matrix computations namely a *global translational dependence* of the computations set. In the common case this peculiarity resides in the existence of a subset $\mathcal{P}_{int}^{(p)} = \{p_1, \ldots, p_l\} \subset \mathcal{P}_{int}$, such as

1. all computations from the subset $\mathcal{P}_{int}^{(p)}$ have the input dependence on the same output variable of the computation at some point $p \in \mathcal{P}_{in} \cup \mathcal{P}_{int}$;
2. the number of elements of the subset $\mathcal{P}_{int}^{(p)}$ lays in a proportion with a size of problem solved (i.e., size of input data).

The computations subset $\mathcal{P}_{int}^{(p)}$ forms a *domain of affection* of the computation at the point p, i.e., for any point $p_j \in \mathcal{P}_{int}^{(p)}$ $(j \in \{1, \ldots, l\})$ the following equality is fulfilled

$$p_j - \Theta_i(p_j) = A_i \cdot p_j + b_i = p.$$

The global translational dependence is costly when actually constructed in VLSI or mapped on the existing massive-parallel computer. Thus, it has to be eliminated by reducing the *non-uniform* system (3) to a *uniform* system of the form like the following

$$\begin{cases} x_1(p) \leftarrow \ \textbf{if } \Theta_1(p) = \mp e_1 \textbf{ then } y(p - \Theta_1) \textbf{ else } x_1(p \pm e_1)\ , \\ x_2(p) \leftarrow \ \textbf{if } \Theta_2(p) = \mp e_2 \textbf{ then } y(p - \Theta_2) \textbf{ else } x_2(p \pm e_2)\ , \\ x_3(p) \leftarrow y(p - \Theta_3), \\ y(p) \leftarrow f(x_1(p), x_2(p), x_3(p)), \end{cases} \tag{4}$$

where $p \in \mathcal{P}_{int}$; $e_1, e_2, (e_3 = \Theta_3)$ are *local data dependence vectors* (LDD-vectors), which satisfy the condition $|e_i| = 1$ (condition of locality of the dependence of the computation p on the neighboring (in the sense of the $\mathcal{I}$ metrics) computation $(p - e_i)$). The following theorem is applied

Theorem 1 (pipelining). *For the $\mathcal{I} = \mathsf{Z}^3$ case the equations system (3) can be reduced to a system like (4) iff*

$$\mathsf{dim\,ker}\, A_1,\ \mathsf{dim\,ker}\, A_2 \in \{1, 2\},$$

where dim *stands for the dimension of the space that follows,* ker A *designates a kernel space of the linear transformation A ($v \in \mathsf{ker}\, A \Leftrightarrow A \cdot v = \vec{0}$).*

A constructive uniformization procedure was given in the proof of this theorem [16].

A *local data dependencies graph* (LDD-graph) corresponding to the equations system (4) has the form

$$\mathcal{G}^* = (\mathcal{P}_{int}, E \subseteq \{\pm e_1, \pm e_2, e_3\}),$$

and being supplemented by the input-output computation-nodes is used by S4CAD as the source algorithm representation.

2.1. Space-time analysis

2.1.1. Time scheduling

The time scheduling is given by a *timing* (step) *function*

$$\mathsf{step}(p) : \mathcal{P}_{int} \to \mathsf{Z},$$

which makes a correlation between the nodes from $\mathcal{P}_{int}$ and *execution steps*, i.e., this function sets a complete time-ordering for the partially ordered computations set.

The function $\mathsf{step}(p)$ is found in the linear form [13, 5]

$$\mathsf{step}(p) = \alpha^{\mathsf{T}} \cdot p + \beta,$$

where $p \in \mathcal{P}_{int} \subset \mathsf{Z}^3$, $\alpha \in \mathsf{Z}^3$, $\beta \in \mathsf{Z}$.

Having the definition of the timing function $\mathsf{step}(p)$ the notion of a *flow velocity vector* of a variable v along a direction e_v is introduced [5]:

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

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

If we assume $\mathsf{step}(p^{min}) = 0$ for all minimal points of the partially ordered computations set, an extension of the input-output computations domains, necessary to obtain correct allocation of the input-output data flows on the processing space (see below), is done by the following formulae:

- for the input data

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

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

- for the output data

$$q_{out} = q_0 + (\mathsf{step}(p^{max}) - \mathsf{step}(q_0) + 1)\mathsf{flow}(u),$$

  where $q_0 \in \mathcal{P}_{out}$, u is the output variable of the algorithm corresponding to the computation at the point q_0, $p^{max} \in \mathcal{P}_{int}$ is one of the maximal points of the partially ordered computations set, $\mathsf{step}(q_{out}) = \mathsf{step}(p^{max}) + 1$.

2.1.2. Spatial allocation

The spatial allocation of the computations set given in the space $\mathcal{I} = \mathsf{Z}^n$ on the space $\mathcal{S} = \mathsf{Z}^{n-1}$ with the metrics induced by the one of the $\mathcal{I}$ is accomplished by an *allocation function*

$$\mathsf{place}(p) : \mathcal{I} \rightarrow \mathcal{S}.$$

This function gives for each computation-node $p \in \mathcal{P}$

- either (when $p \in \mathcal{P}_{int}$) $\mathcal{S}$-coordinates of a *processing element* (PE), which will execute the computation of the node p at the $\mathsf{step}(p)$;
- or (when $p \in \mathcal{P}_{in} \cup \mathcal{P}_{out}$) coordinates of the point where the element p of the input-output data will be allocated trough the recurrent steps.

A linear form of the allocation function is used:

$$\mathsf{place}(p) = \Lambda_\eta \cdot p,$$

where Λ_η is the $[(n-1)\times n]$-matrix of a linear transformation corresponding to a *projection vector* $\eta \in \ker \Lambda_\eta$ and which has the $\mathsf{rank}\, \Lambda_\eta = n - 1$.

The set $\mathcal{S}_\eta = \{\Lambda_\eta \cdot p \mid p \in \mathcal{P}_{int}\} \subset \mathcal{S}$ corresponding to the projection vector η is called a *processor array* (PE-array). Projection vector η is considered *admissible* only if the scalar product

$$(\alpha, \eta) \neq 0,$$

where α is the coefficients vector of the linear form of the timing function.

Data Flow Allocation If a variable v propagating trough the space $\mathcal{I} = \mathsf{Z}^n$ along a direction e_v^n and having a velocity $\mathsf{flow}^n(v)$ is mapped into the space $\mathcal{S} = \mathsf{Z}^{n-1}$, it will propagate along the direction $e_v^{n-1} = \Lambda_\eta \cdot e_v^n$ and have the velocity $\mathsf{flow}^{n-1}(v) = \Lambda_\eta \cdot \mathsf{flow}^n(v)$. When $\mathsf{flow}^{n-1}(v) = \vec{0}$, variable v is said to be *stationary*.

Variables having the same velocity-direction vector form a *data flow*.

In the case when non-stationary variable is going to be allocated on any PE it is expected to *move* the data flow having this variable *out* from the PE-array, certainly increasing a task processing time.

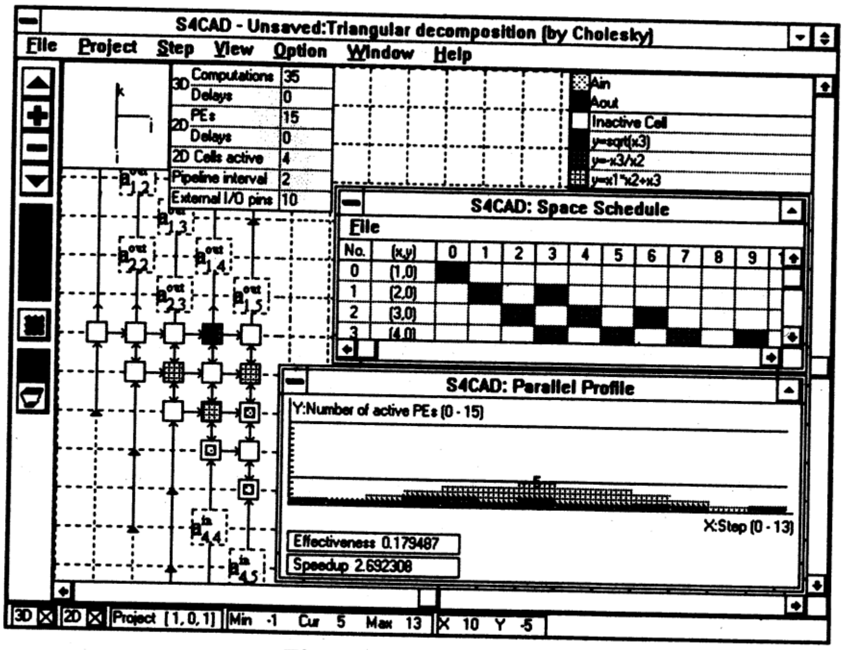

Figure 1. S4CAD Screen

3. S4CAD overview

S4CAD runs under the graphical operating environment MICROSOFT WINDOWS 3. S4CAD gives to a designer convenient control means and presents the information graphically for better understanding.

At the current realization stage parametrized local dependencies graph, timing function in its linear form and data flow vectors (direction-velocity ones) form the source specification of an algorithm for the S4CAD.

S4CAD allows user the following: (see Figure 1)

- interactively choose an algorithm from a library and adjust its input parameters (problem size);
- obtain and view on the screen the set of all admissible *projects* of the systolic architecture for the given algorithm;
- execute step-by-step simulation of the data processing and communications for any chosen project;
- get a visual information about the following project characteristics:
  - structure topology,
  - number of computations and processing time,

- number of PEs and delays,
- number of external links,
- data flows formats,
- number of active PEs on the current processing step,
- PEs specialization,
- executed operations types etc.;

- get a parallelization time profile (histogram), estimate the effectiveness of the resource use and the speedup of a parallel execution over the sequential one;

- get the different space-time schedulings of the computations set of the algorithm;

- make a use of a number of the service functions: get a help, information on the algorithm chosen, change scale, placement and style of the view, save/restore data in files, get a printed copy of the design and so on;

- enrich the library by new algorithms.

The parametrized algorithms now contained in the S4CAD library are the following: banded matrices product, the Gaussian LU-decomposition, Cholessky's triangularization, matrix inversion by Gauss-Jordan, transitive closure algorithm by Warshall, shortest path problem by Floyd, finding minimal spanning tree algorithm by Maggs-Plotkin.

One of the main goals of the existing version of S4CAD is a help to the designer to choose an optimal project from a set of alternative ones. Some of generally applied criteria are the following:

- problem solving time;

- number of PEs and delays;

- PEs specialization (PE doing one predefined operation firstly, simpler when constructed and secondly, it does not require control over itself);

- structure topology (in certain cases the orthogonal dependencies may be preferable to the hexagonal ones);

- data flows formats (in the case when different systolic structures combination is required the data flows formats may be essential);

- number of the external links (number of the external input-output pins is essential for VLSI-implementation);
- structure modularity (the structure is said to be *modular* when one may easily combine modules to solve a problem of a grater size, for example, four modules for matrix multiplication of the order n may form a device for matrix multiplication of the order $2n$).

4. An example: transitive closure algorithm

Let's illustrate the design of systolic structures on the example of the transitive closure algorithm.

4.1. Problem constitution

An incidence matrix $A_{in} = [a_{ij}^{in}]_{n\times n}$ of a graph $\mathcal{U}$ of n nodes is given. It is demanded to find out matrix $A_{out} = [a_{ij}^{out}]_{n\times n}$ possessing the following property

$$a_{ij}^{out} = \mathsf{true} \Leftrightarrow \exists\, \omega_{i\to j},$$

where $\omega_{i\to j}$ is a path begining from the node i and ending in the node j of the given graph $\mathcal{U}$.

The following Warshall's algorithm essentially depends on the property of the incidence matrix of a graph: $a_{ij}^{in} = \mathsf{true}$ when $i = j$.

4.2. Affine recurrent equations[2]

// input computations
for all $1 \le i, j \le n$ **do** $a(i, j, 0) := a_{ij}^{in}$;

// internal computations
for $k := 1$ **to** n **do**
  begin
    $a(n+k, n+k, k) := a(k, k, k) := a(n+k, k, k) := \mathsf{true}$;
    for all $k+1 \le i \le n+k-1$ **do**
      $a(i, n+k, k) := a(i, k, k) := \mathsf{true}$;
    for all $k+1 \le j \le n+k-1$ **do**
    begin
      for all $k+1 \le i \le n+k-1$ **do**
        $a(i, j, k) := a(i, j, k-1) \wedge a(i, k, k) \vee a(k, j, k)$;

[2] The source script is given in the form of nested loops [17].

$a(k,j,k) := a(k+1, j+k, k-1);$
$a(n+k,j,k) := \text{true};$
end
end

// output computations
for all $n \le i,j \le 2n$ **do** $a_{ij}^{out} := a(i,j,n);$

It is evident that for $p = [i,j,k]^{\top}$ $\Theta_3 = [i,j,k]^{\top} - [i,j,k-1]^{\top} = [0,0,1]^{\top}$, $\Theta_2(p) = [i,j,k]^{\top} - [i,k,k]^{\top} = [0, j-k, 0]^{\top}$, $\Theta_1(p) = [i,j,k]^{\top} - [k,j,k]^{\top} = [i-k, 0, 0]^{\top}$,

$$A_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, b_3 = -\Theta_3, A_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}, A_1 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

$$b_1 = b_2 = \vec{0}.$$

According to $\dim\ker A_1 = \dim\ker A_2 = 1$ we may apply the pipelining Theorem and reduce the affine equations to uniform recurrent equations.

4.3. Uniform recurrent equations

// input computations
$\{\ a(p) \ = \ a_{ij}^{in}; \quad p \in \mathcal{P}_{in}(A_{in})$

// internal computations

$$\begin{cases} x_1(p) &= \textbf{if } i < n+k \ \& \ i > k \textbf{ then} \\ & \quad \{\textbf{if } i = k+1 \textbf{ then } a(p-e_1) \\ & \quad\ \ \textbf{else } x_1(p-e_1)\}; \\ x_2(p) &= \textbf{if } i > k \ \& \ j > k \textbf{ then} \\ & \quad \{\textbf{if } j = k+1 \textbf{ then } \ a(p-e_2) \\ & \quad\ \ \textbf{else } x_2(p-e_2)\}; \\ x_3(p) &= \textbf{if } i < n+k \ \& \ j < n+k \textbf{ then } a(p-e_3); \\ a(p) &= \textbf{if } i = j = k \textbf{ then true} \qquad\qquad\qquad\qquad p \in \mathcal{P}_{int} \\ & \quad \textbf{else if } i = k \textbf{ then } x_3(p) \\ & \quad \textbf{else if } i = n+k \ \& \ j = k \textbf{ then } x_1(p) \\ & \quad \textbf{else if } j = n+k \textbf{ then } x_2 \\ & \quad \textbf{else if } j = k \textbf{ then } x_3(p) \text{ when } x_1(p) \\ & \quad \textbf{else if } i = n+k \textbf{ then } x_1(p) \text{ when } x_2(p) \\ & \quad \textbf{else if } i = j \textbf{ then } x_3(p) \text{ when } x_1(p) \text{ with } x_2(p) \\ & \quad \textbf{else } x_3(p) \wedge x_1(p) \vee x_2(p) \end{cases}$$

// output computations
$\{\ a_{ij}^{out} \ = \ a(p); \quad p \in \mathcal{P}_{out}(A_{out}),$

where $e_1 = [1,0,0]^\top, e_2 = [0,1,0]^\top, e_3 = [0,0,1]^\top$ are LDD-vectors; $\mathcal{P}_{in}(A_{in}) = \{(i,j,0)^\top \mid 1 \le i,j \le n\}$, $\mathcal{P}_{out}(A_{out}) = \{(i{+}n, j{+}n, n{+}1)^\top \mid 1 \le i,j \le n\}$, $\mathcal{P}_{int} = \{(i,j,k)^\top \mid 1 \le k \le n,\ k \le i,j \le n+k,\ (i \ne k) \vee (j \ne n+k)\}$, operations a **when** b and a **when** b **with** c mean the unit delay of the variable a synchronized by the presence of values in the variables b and c at the point p.

The data flow vectors are: $\mathsf{flow}(A_{in}) = \mathsf{flow}(A_{out}) = [0,0,1]^\top = e_3$. The minimal form of the timing function is: $\mathsf{step}(p) = i + j + k - 3$, i.e., $\alpha = [1,1,1]^\top$, $\beta = -3$. The minimal point is $p^{min} = (1,1,1)^\top$, the maximal point is $p^{max} = (2n, 2n, n)^\top$, i.e., the least processing time is $\mathsf{step}(p^{max}) = 5n - 3$.

4.4. Projective solutions analysis

$\eta = [1,0,0]^\top$ (Figure 2) This project is characterized by the least admissible processing time: $5n-3$, number of PEs: $(n-1)^2$, delays: $2n$, data pipelining period: $|(\alpha,\eta)| = 1$, number of external links: $2n$.

$\eta = [1,1,1]^\top$ (Figure 3) This project has the least among others number of PEs: $(n-1)^2 - n = n^2 - n + 1$, delays: $2n+1$, processing time: $7n-4$ ($2n$ because of data flows movement out), data pipelining period: $|(\alpha,\eta)| = 3$, number of external links: $2(2n-1) = 4n-2$.

$\eta = [1,1,0]^\top$ (Figure 4) This project is characterized by the data processing time: $5n-3$, number of PEs: $(2(2n-1)-3)n = 4n^2 - 5n$, delays: $4n$, external links: $4n-2$, data pipelining period: $|(\alpha,\eta)| = 2$

5. Conclusion

Thus, the main properties of the existing S4CAD realization were considered above. Let's mark the development trends.

1. Parse a high-level recurrences description to the data dependencies graph (for example, the ALPHA language [4]).
2. Automatical uniformization including one that deals with a limited broadcast [15].
3. Analysis and simulation of the computational and communicational processes of the source data dependencies graph (for instance, aiming at the optical computing and three-dimensional VLSI-circuits).

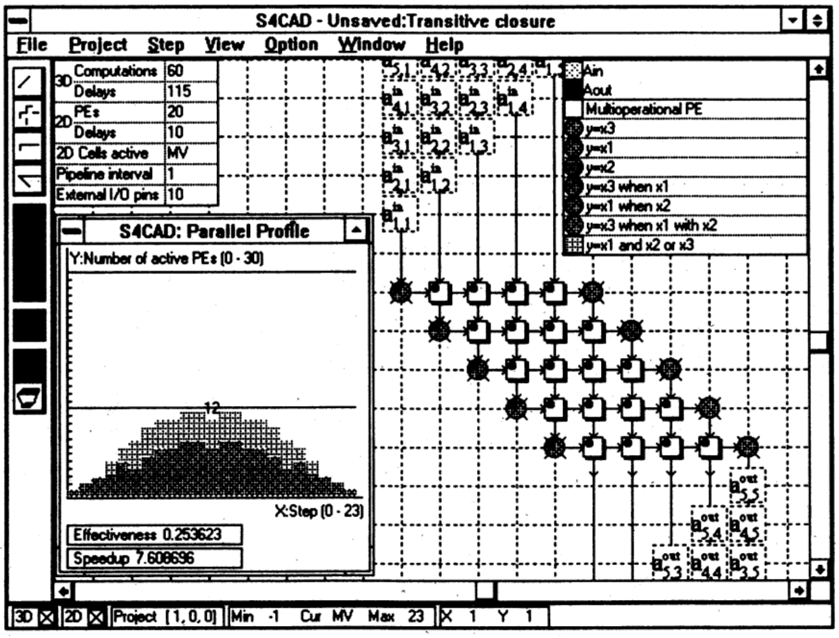

Figure 2. Transitive closure algorithm, project $\eta = [1, 0, 0]^{\mathsf{T}}$

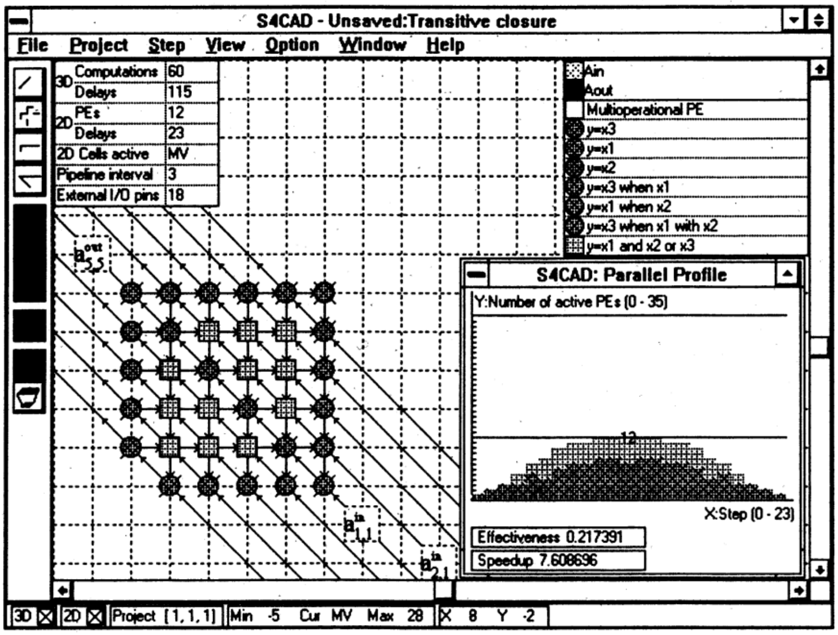

Figure 3. Transitive closure algorithm, project $\eta = [1, 1, 1]^{\mathsf{T}}$

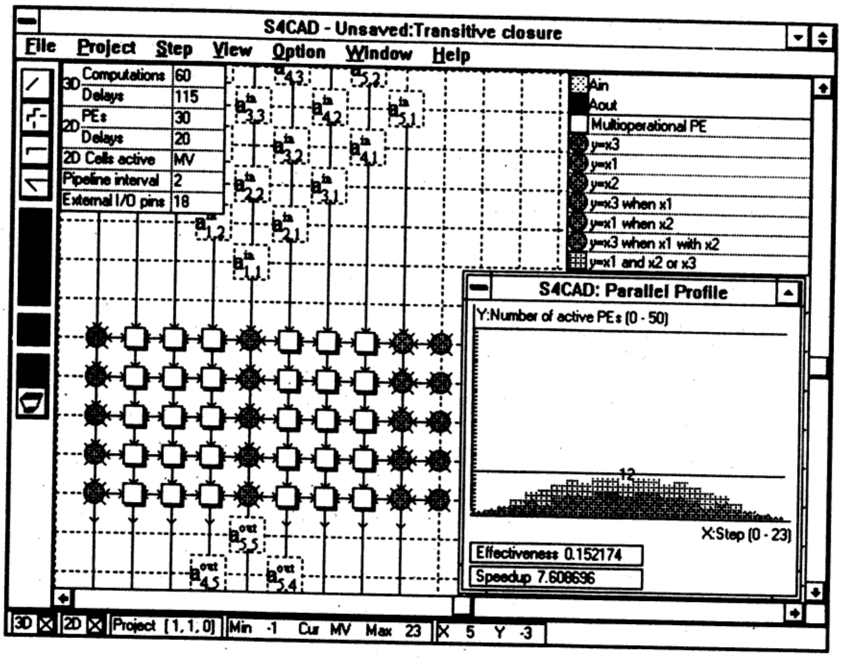

Figure 4. Transitive closure algorithm, project $\eta = [1, 1, 0]^{\mathrm{T}}$

4. Wavefront analysis and modeling with a vast consideration of computation and communication costs.
5. Automatic synthesis of control signals for PE-arrays [14, 19].
6. Computations arrangement on the systolic structures with a bounded number of PEs (i.e., *partitioning* [3]).
7. Application of different allocation methods including non-linear ones [1].
8. (a) Use of S4CAD in the education for formal synthesis and analysis of highly-parallel fine-grain algorithms and structures.
   (b) Interaction with silicon and parallel language compilers.
   (c) Application of S4CAD for mapping some algorithms onto the existing massive-parallel computers, which allow realization of the systolic computations [8].

References

[1] A. Benani, Y. Robert, Space-time-minimal systolic arrays for Gaussian elimination and the algebraic path problem, Parallel Computing, No.15, 1990, 211-225.

[2] W. Burleson, B. Jang, ARREST: an interactive graphic analysis tool for VLSI arrays, Proc. Int. Conf. on Application Specefic Array Processors. Aug. 4-7, 1992, Berkley, Calif. IEEE Comp. Society, 1992, 149-162.

[3] A. Darte, Regular partitioning for synthesing fixed-size systolic arrays, Techical Report 91-10, Laboratoire LIP-IMAG, Ecole Normale Superiore de Lyon, April, 1991.

[4] C. Dezan, H. Le Verge, P. Quinton, Y. Saouter, Alpha du Centaur environment, P.Quinton and Y.Robert eds., Int. Workshop Algorithms and Parallel VLSI Architectures II, Bonas, France, June, 1991, 325-334.

[5] C.-H. Huang, C.Lengauer, The derivation of systolic implementations of programs, Acta Informatica 24, 1987, 595-632.

[6] P.P. Hou, R.M. Ownes, M.J. Irwin, DECOMPOSER: a synthesizer for systolic systems, Proc. 25th ACM/IEEE Int. Conf. Automatic Design, Anaheim, June 12-15, 1988, N.Y., 1988, 650-653.

[7] S.Y. Kung, S.N. Jean Jack, Array compiler design for VLSI/WSI systems, Proc. of Int. Conf. on Systolic Arrays, Killarney, Co.Kerry, Ireland, 1989, 665-679.

[8] C. Lengauer, M. Barnett, D.G. Hudson, Towards systolizing compilation, Distributed Computing, Vol.5, No.1, 1991, 7-24.

[9] D.I. Moldavan, ADVIS: a software package for the design of systolic arrays, IEEE Trans. on Computer-Aided Design, Vol.6, No.1, 1987, 33-40.

[10] C. Mongenet, G.R. Perrin, Synthesis of systolic arrays for inductive problems, Proc. of Int. Conf. Parallel Architectures and Languages Europe (PARLE'89), Lecture Notes in Computer Science, No.365, Springer-Verlag, 1989, 260-277.

[11] E.T. Omtizigt, SYSTARS: a CAD tool for the synthesis and analysis of VLSI systolic/wavefront arrays, Proc. Int. Conf. Systolic Arrays, San Diego, Calif. May 25-27, 1988. Washington D.C., 1988, 383-391.

[12] P. Quinton, P. Frison, P. Gachet, Synthesing systolic arrays using DIASTOL, VLSI Signal Processing II, IEEE Press, 1986, 93-105.

[13] P. Quinton, The systematic design of systolic arrays, IRISA Internal Report, No.216, INRIA, 1983.

[14] S.V. Rajopadhye, Synthesing systolic arrays with control signals from recurrence equations, Distributed Computing, No.3, 1989, 88-105.

[15] T. Risset, Y. Robert, Synthesis of processor arrays for the algebraic path problem: unifying old results and deriving new architectures, Parallel Processing Letters, Vol.1, No.1, 1991, 19-28.

[16] S.G. Sedukhin, Design and analysis of systolic algorithms and structures, Russian Software, No.2, 1991, 20-40 (in Russian).

[17] S.G. Sedukhin, Design and analysis of systolic algorithms for the algebraic path problem, Computers and Artificial Intelegence, Vol.11, No.3, 1992, 269-292.

[18] V.van Dongen, M. Petit, PRESAGE: a tool for the parallelization of nested loop programs. Formal VLSI Specification and Synthesis, VLSI Design Methods I. North-Holland, 1990, 341-359.

[19] J. Xue, C. Lengauer, The systematic derivation of control signals for systolic arrays, Report of the Depatment of Computer Science of the University of Edinburg, ECS-LFCS-91-152, 1991.