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 analyze 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 $I = \mathbb{Z}^n$ and the admissible range of this variables defines an internal computations domain $\mathcal{P}_{\text{int}}$, 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 $\mathcal{P}_{\text{in}}$ and, respectively, final assignments form an output computations domain $\mathcal{P}_{\text{out}}$. The set

$$\mathcal{P} = \mathcal{P}_{\text{in}} \cup \mathcal{P}_{\text{int}} \cup \mathcal{P}_{\text{out}}$$
is called *computations domain* of the algorithm.

The original algorithm representation is taken in the form of a system of *affine recurrent equations*\(^1\)

\[ y(p_0) = f(y(p_0 - \Theta_1(p_0)), \ldots, y(p_0 - \Theta_m(p_0))), \]

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 \(p \in \mathcal{P}; \Theta_i(p_0) \in \mathcal{I}, i \in \{1, \ldots, m\}\) 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

\[ p_i = p_0 - \Theta_i(p_0) = A_i \cdot p_0 + b_i, \]

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

The \(\rho(\bar{x}, \bar{y}) = \max_i |x_i - y_i|\) 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., \(\mathcal{P} \subset \mathcal{I} \subseteq \mathbb{Z}^3 = \{(i,j,k) \in \mathbb{Z}^3 \mid i,j,k \in \mathbb{Z}\},\) where \(\mathbb{Z}\) 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 \(\Theta_3(p) = [0,0,1]^T\) is used for *reverse* and \(\Theta_3(p) = [0,0,-1]^T\) for *normal* recurrence. Below the assumption \(\mathcal{I} = \mathbb{Z}^3\) is made, the extension to \(\mathcal{I} = \mathbb{Z}^n\) is considered trivial.

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

\[
\left\{
\begin{array}{l}
x_1(p) = y(p - \Theta_1(p)), \\
x_2(p) = y(p - \Theta_2(p)), \\
x_3(p) = y(p - \Theta_3), \\
y(p) = f(x_1(p), x_2(p), x_3(p)),
\end{array}
\right.
\]

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 \(p\) if 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 $G$ is supplemented by the input-output nodes with the necessary arcs.

The equations system (3) (and as a more illustrative form DDD-graph $G$) 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 $P_{int}^{(p)} = \{p_1, \ldots, p_l\} \subset P_{int}$, such as

1. all computations from the subset $P_{int}^{(p)}$ have the input dependence on the same output variable of the computation at some point $p \in P_{in} \cup P_{int}$;

2. the number of elements of the subset $P_{int}^{(p)}$ lays in a proportion with a size of problem solved (i.e., size of input data).

The computations subset $P_{int}^{(p)}$ forms a domain of affection of the computation at the point $p$, i.e., for any point $p_j \in P_{int}^{(p)} (j \in \{1, \ldots, l\})$ 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

$$
\begin{align*}
  x_1(p) & \leftarrow \text{if } \Theta_1(p) = \mp e_1 \text{ then } y(p - \Theta_1) \text{ else } x_1(p \pm e_1), \\
  x_2(p) & \leftarrow \text{if } \Theta_2(p) = \mp e_2 \text{ then } y(p - \Theta_2) \text{ else } x_2(p \pm e_2), \\
  x_3(p) & \leftarrow y(p - \Theta_3), \\
  y(p) & \leftarrow f(x_1(p), x_2(p), x_3(p)),
\end{align*}
$$

(4)

where $p \in P_{int}$; $e_1, e_2, (e_3 = \Theta_3)$ 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 $I$ metrics) computation $(p - e_i)$). The following theorem is applied

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

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

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

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

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

\[ \text{step}(p): \mathcal{P}_{\text{int}} \rightarrow \mathbb{Z} , \]

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

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

\[ \text{step}(p) = \alpha^T \cdot p + \beta , \]

where \( p \in \mathcal{P}_{\text{int}} \subset \mathbb{Z}^3 , \alpha \in \mathbb{Z}^3 , \beta \in \mathbb{Z} \).

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

\[ \text{flow}(v) = \frac{q - p}{\text{step}(q) - \text{step}(p)} , \]

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

If we assume \( \text{step}(p^{\text{min}}) = 0 \) for all minimal points of the partially ordered 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

\[ p_{\text{in}} = p_0 - (\text{step}(p_0) + 1)\text{flow}(v) , \]

where \( p_0 \in \mathcal{P}_{\text{in}} \), \( v \) is the input variable of the algorithm corresponding to the computation at the point \( p_0 \), \( \text{step}(p_{\text{in}}) = -1 \);
• for the output data
  \[ q_{out} = q_0 + (\text{step}(p_{max}) - \text{step}(q_0) + 1)\text{flow}(u), \]
  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, \( \text{step}(q_{out}) = \text{step}(p_{max}) + 1 \).

2.1.2. Spatial allocation

The spatial allocation of the computations set given in the space \( \mathcal{I} = \mathbb{Z}^n \) on the space \( S = \mathbb{Z}^{n-1} \) with the metrics induced by the one of the \( \mathcal{I} \) is accomplished by an allocation function
  \[ \text{place}(p) : \mathcal{I} \rightarrow S. \]

This function gives for each computation-node \( p \in \mathcal{P} \):
- either (when \( p \in \mathcal{P}_{int} \)) \( S \)-coordinates of a processing element (PE), which will execute the computation of the node \( p \) at the \( \text{step}(p) \);
- 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 through the recurrent steps.

A linear form of the allocation function is used:
  \[ \text{place}(p) = \Lambda_\eta \cdot p, \]
  where \( \Lambda_\eta \) 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 rank \( \Lambda_\eta = n - 1 \).

The set \( \mathcal{S}_\eta = \{ \Lambda_\eta \cdot p \mid p \in \mathcal{P}_{int} \} \subset S \) corresponding to the projection vector \( \eta \) is called a processor array (PE-array). Projection vector \( \eta \) is considered admissible only if the scalar product
  \[ (\alpha, \eta) \neq 0, \]
  where \( \alpha \) is the coefficients vector of the linear form of the timing function.

Data Flow Allocation If a variable \( v \) propagating through the space \( \mathcal{I} = \mathbb{Z}^n \) along a direction \( e_\eta^n \) and having a velocity \( \text{flow}^n(v) \) is mapped into the space \( S = \mathbb{Z}^{n-1} \), it will propagate along the direction \( e_\eta^{n-1} = \Lambda_\eta \cdot e_\eta^n \) and have the velocity \( \text{flow}^{n-1}(v) = \Lambda_\eta \cdot \text{flow}^n(v) \). When \( \text{flow}^{n-1}(v) = 0 \), 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.
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, Cholesky’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 greater 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 \( U \) of \( n \) nodes is given. It is demanded to find out matrix \( A_{out} = [a_{ij}^{out}]_{n \times n} \) possessing the following property

\[
a_{ij}^{out} = \text{true} \iff \exists \omega_{i \rightarrow j},
\]

where \( \omega_{i \rightarrow j} \) is a path beginning from the node \( i \) and ending in the node \( j \) of the given graph \( U \).

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

4.2. Affine recurrent equations\(^{2}\)

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

// internal computations
for \( k := 1 \) to \( n \) do

begin
\( a(n + k, n + k, k) := a(k, k, k) := a(n + k, k, k) := \text{true}; \)
for all \( k + 1 \leq i \leq n + k - 1 \) do
\( a(i, n + k, k) := a(i, k, k) := \text{true}; \)
for all \( k + 1 \leq j \leq n + k - 1 \) do

begin
for all \( k + 1 \leq i \leq n + k - 1 \) do
\( a(i, j, k) := a(i, j, k - 1) \land a(i, k, k) \lor a(k, j, k); \)

\(^{2}\) The source script is given in the form of nested loops [17].
\begin{align*}
a(k, j, k) &:= a(k + 1, j + k, k - 1); \\
a(n + k, j, k) &:= \text{true}; \\
\end{align*}

end

// output computations
for all \( n \leq i, j \leq 2n \) do \( a^{\text{out}}_{ij} := a(i, j, n); \)

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

\[
A_3 = \begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
\end{bmatrix},
\]

\[
b_3 = -\Theta_3, A_2 = \begin{bmatrix}
1 & 0 & 0 \\
0 & 0 & 1 \\
0 & 0 & 1 \\
\end{bmatrix},
\]

\[
A_1 = \begin{bmatrix}
0 & 0 & 1 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
\end{bmatrix},
\]

\[
b_1 = b_2 = 0.
\]

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^{\text{in}}_{ij}; \ p \in P_{in}(A_{in})
}

// internal computations
{ x_1(p) = \begin{cases} 
\text{if } i < n + k \&\& i > k \text{ then} \\
\{ \text{if } i = k + 1 \text{ then } a(p - e_1) \\
\text{else } x_1(p - e_1) \}; 
\end{cases} \quad p \in P_{int} \\
\text{if i = k then x_3(p)}
\text{else if i = n + k \&\& j = k then x_1(p)}
\text{else if j = n + k then x_2}
\text{else if j = k then x_3(p) when x_1(p)}
\text{else if i = n + k then x_1(p) when x_2(p)}
\text{else if i = j then x_3(p) when x_1(p) with x_2(p)}
\text{else x_3(p) \& x_1(p) \lor x_2(p)} }
// output computations
\{ \ a_{ij}^{ou} = \ a(p); \ p \in P_{out}(A_{out}), \}

where \ e_1 = [1, 0, 0]^T, e_2 = [0, 1, 0]^T, e_3 = [0, 0, 1]^T \ are \ LDD-vectors;
\ P_{in}(A_{in}) = \{(i,j,0)^T | 1 \leq i, j \leq n\}, \ P_{out}(A_{out}) = \{(i+n+j+n,n+1)^T | 1 \leq i, j \leq n\}, \ P_{int} = \{(i,j,k)^T | 1 \leq k \leq n, \ k \leq i, j \leq n + k, \ (i \neq k) \lor (j \neq 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. \n
The data flow vectors are: \ flow(A_{in}) = flow(A_{out}) = [0, 0, 1]^T = e_3. \n
The minimal form of the timing function is: \ step(p) = i + j + k - 3, \ i.e., \ \alpha = [1,1,1]^T, \ \beta = -3. \ The \ minimal \ point \ is \ p^{min} = (1,1,1)^T, \ the \ maximal \ point \ is \ p^{max} = (2n,2n,n)^T, \ i.e., \ the \ least \ processing \ time \ is \ step(p^{max}) = 5n - 3. \n
4.4. Projective solutions analysis

\eta = [1,0,0]^T (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]^T (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]^T (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 ALPH A 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 $\eta = [1, 0, 0]^T$

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


