Theoretical Computer Science 71 (1990) 47-77
North-Holland

SYNTHESIS, STRUCTURE AND POWER OF SYSTOLIC COMPUTATIONS

Jozef GRUSKA

Institute of Technical Cybernetics, Slovak Academy of Sciences, Dúbravská 9, 842 37 Bratislava, Czechoslovakia

Abstract. A variety of problems related to systolic architectures, systems, models and computations are discussed. The emphases are on theoretical problems of a broader interest. Main motivations and interesting/important applications are also presented. The first part is devoted to problems related to synthesis, transformations and simulations of systolic systems and architectures. In the second part, the power and structure of tree and linear array computations are studied in detail. The goal is to survey main research directions, problems, methods and techniques in not too formal a way.

1. Introduction

Systolic architecture has been one of the most attractive ideas in computer architecture so far. It is very appealing from the design point of view, because it is based on repetitions of simple processors, and on regularity, modularity and simplicity of interconnections. Moreover, many systolic systems can be designed using only very few types of processors, and also the repeated use of each input data in systolic systems significantly minimizes interaction with the memory of the host computing environment. All these are also reasons why systolic architecture is so suitable to make full use of the great potential of VLSI technology. Due to its simplicity, modularity and repeatability, systolic architecture also offers transparent, understandable and manageable, but still quite powerful, parallelism. Intricacy of data and communication flow in many systolic systems, offer the magic so useful and necessary to attract larger groups of designers to the idea. At the same time, a large variety of interesting theoretical problems of fundamental importance, and broader implications, have arisen in connection with the synthesis, analysis, and implementation of systolic and related systems and computations.

Systolic systems can be seen as an interesting and useful modification (simplification mostly), of the cellular automata concept of von Neumann, with more emphasis on regularity and transparency of data and computation flow. New models and new problems have been motivated mainly by advances in technology. Similarity with cellular automata has immediately brought into use a whole bunch of theoretical techniques to study systolic architectures, systems and computations. On the other hand, systolic architecture is a natural modification (generalization mostly) of pipelining—the architectural concept that has contributed so much in recent years to the significant increase in performance of modern computers. The creation of a quite general model of a practically important concept has been once again an important tool leading to a new quality—this time in computer architecture and supercomputing. This has, in turn, brought a new set of powerful formal methods into computer architecture.

Two of the most powerful recent concepts in computer architecture, systolic systems by H. T. Kung and RISC architecture by J. Cocke, have quite similar genesis. Both authors arrived at their ideas which are of major importance for computer architecture, by doing mainly theoretical research in quite remote areas of computer science and having in common mainly a long term search for improving efficiency.

The first and nowadays seminal systolic systems by Kung and Leiserson—for matrix multiplication and LR-decomposition—appeared in 1978. Since then various systolic systems have been designed. Several systolic system implementation patents have been taken in various countries.

Most of the systolic systems have been designed using ad hoc methods and in many cases very similar systolic systems have been designed for seemingly quite different problems. This has naturally led to intensive research oriented to developping systematic and sufficiently automatizable methods to synthesise systolic systems from high level specifications. Significant progress has been achieved in this direction and several systolic system design methodologies are also discussed in this paper. The method due to Ibarra and his coworkers [49, 48] deserves special attention. Inspite of being theory oriented and inspired, this method is very powerful and allows the design of systolic systems of various architectures to be reduced to the design of sequential programs for various simple sequential machines—a long time desire in the area of parallel computations. In addition, various theoretically justified, systolic system transformation techniques and systolic architecture simulation techniques have been developed. They also contribute significantly to the improvement and efficiency of systolic system design methodologies [71, 8].

The path from systolic systems, represented by attractive but still very high level abstract networks, to real and efficient implementations is long, indirect, and far from easy. In order to obtain really useful VLSI implementations, various tradeoffs and design modifications have to be considered [34, 29, 68, 85], and really very high density integrated circuits are required. All this can be seen on perhaps the main project in this area, the warp machine at CMU. This machine consists of a linear array of programmable processors that have been designed in such a way that the whole array can implement efficiently various systolic systems especially for vision and signal processing computations. This project indicates especially well how long and complicated is the way from a simple though powerful idea to a really successful machine. Implementation problems have also led to some interesting theoretical problems.

VLSI implementations are not the only way to make use of the advantages of systolic systems. For several reasons they are also very suitable for effective simulations on current multiprocessor systems, for example multitransputer systems. This has led to attempts to develop systolic programming methodologies, programming languages and environments [32].

There have also been numerous attempts to develop and study various abstract models of systolic systems and computations and this will be our main interest in this paper. The models are of various degrees of abstraction and often far more general than a naive view of systolic systems, deduced from main examples, would suggest. The range of problems to be investigated for a particular model is partly determined by the generality and the abstraction of the model, but mainly by our desire to improve our insight and knowledge concerning synthesis, behavior and analysis of systolic architectures, systems and computations and to improve our methodology for dealing with them.

The concept of topological transformation, as a formalization of time-space transformations, is a very useful tool to study various problems in the area of systolic computations, especially in the area of simulations of systolic architectures and transformations of systolic systems. Various results useful for the design and synthesis of systolic systems have also been obtained using this tool. Of special importance are results concerning the removal of broadcasting and the replacement of two-way communications by one-way, with minimal time overhead. Two special transformations, retiming and slowdown, are nowadays powerful tools for design of networks. Simulation results presented here concentrate on simulation of one parallel architecture on another one, on emulation of large networks on smaller ones and on a "universal" one.

Parallel automata of several types—mainly array, trellis and tree-like networks of finite automata (often memoryless)—have been the main theoretical models used to study various basic problems concerning systolic architectures, systems and computations: the power of various interconnections, the power of one-way and two-way communications, the power of different communication modes with the environment (input/output) and the power of various types of nonhomogeneity. Some of these networks—two-dimensional arrays, trellises and trees—and some types of inputs are shown in Fig. 1.

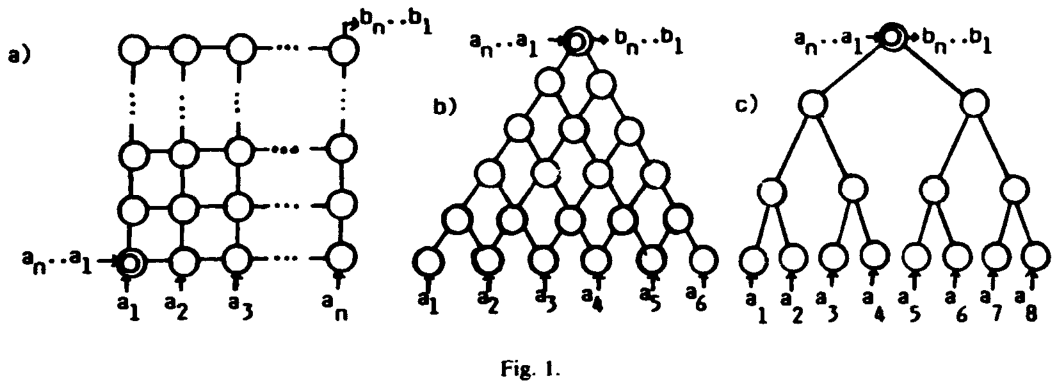

Fig. 1.

Time-space transformations of linear array computations led to the study of other and even more abstract models of systolic computations—two-dimensional infinite words of special types—that are also of a more general interest. Main attention has been devoted so far to the study of infinite two-dimensional words that form so called *generalized Pascal triangles* over arbitrary algebras of the signature (1, 1, 2). They have a rich structure that has allowed various interesting results to be derived concerning linear array computations and their relations with the properties of the underlying algebras.

Cellular automata in general and linear arrays in particular have also been considered as models suitable to study behaviour of complex systems and also as new models of the physical world [84, 90, 92]. Models and results motivated originally by our desire to master parallel computations may also be very useful to obtain deeper results in these new and fundamental investigations.

There are several other surveys on theoretical issues related to systolic computations [37, 43, 87].

2. Systolic systems

There have been several attempts to find a formal definition of systolic systems, for example in [26]. The most fruitful so far is the one given in [70], though at first sight it does not look that way because it is very general. A slight modification of this definition and related theoretical developments and applications are discussed in this section.

Definition 2.1. A *semisystolic system* $S = \langle V, E, \delta, D, \pi \rangle$ is an oriented multigraph, with the set of nodes V, the set of edges E, the data domain D, the processor mapping π that associates with each node $v \in V$ of the indegree $p > 0$, a function $\pi(v): D^p \to D$, and the delay mapping $\delta: E \to N$ (the set of nonnegative integers) such that if δ is extended in a natural way to map also paths of S into N, then $\delta(p) > 0$ for any cycle p. ($\delta(e)$ represents the number of the delay-one registers on the edge e, and the requirement $\delta(p) > 0$ excludes the existence of networks with "unlimited rippling".) The nodes of V of the indegree (outdegree) 0 are called the input (output) nodes or processors of S. If $\delta(e) > 0$ for any $e \in E$, then S is called a *systolic system*.

An informal description of the behaviour of a semisystolic system S is similar to that of a synchronous network working in discrete time. At each moment of the discrete time, each node and also each register output either a value from D or a don't care element, say ('#'). In the case of the register it is the same value as it was at the previous time step on the output of its predecessor (i.e. of the preceding register on the same edge, if there is one, or of the outgoing node of that edge). In the case of the input node it is a value submitted from the environment. Finally, in the case of a node v of the indegree $p > 0$, the corresponding processor outputs the value of the function $\pi(v)$ applied to the arguments produced by its predecessors (registers or nodes) on all ingoing edges at the previous time steps. Moreover, it is assumed that all registers have an "initial value" to start a computation.

It has turned out that in order to be able to deal more precisely with semisystolic systems, especially to formulate precisely the impacts of various transformations, and to develop synthesis techniques, a more rigorous treatment of their semantics is needed.

One way of defining semantics of a semisystolic system S is to associate a "time function" with each node of S [5]. By that we mean an arbitrary partial function from Z (the set of all integers) to D, that is defined only for finitely many negative integers. (The intended interpretation is that the value of the time function associated with a node v, and an argument t, is exactly the value produced by processor in v at the time t.) This can be done as follows.

Let V_I be the set of input nodes of S. Let us first assign a time function τ_v to each input node $v \in V_I$. The semantics of S, with respect to the given input time functions, is then a mapping Φ of nodes of V into time functions, such that $\Phi(v) = \tau_v$ for any $v \in V_I$, and for all other nodes v, $\Phi(v)$ is the minimal fix-point solution of the semantic equations of S. These semantic equations, one for each non-input node of S, relate, in a natural way, through functions associated with node-processors, the time function of each node with time functions of its predecessors on all ingoing edges.

Two types of transformations on semisystolic systems are of special importance: *slowdown transformations* and *retiming transformations.*

If $k \in N^+ = N - \{0\}$, then the k-slowdown transformation of a semisystolic system $S = \langle V, E, \delta, D, \pi\rangle$ results in the semisystolic system $S^{(k)} = \langle V, E, \delta^{(k)}, D, \pi\rangle$, where $\delta^{(k)}(e) = k\delta(e)$, for any $e \in E$. A retiming transformation of S is given by any mapping $r: V \to Z$ such that for any edge $e: u \to v$, $\delta(e) + r(v) - r(u) \geq 0$. The resulting semisystolic system is $S_r = \langle V, E, \delta_r, D, \pi\rangle$, where $\delta_r(e) = \delta(e) + r(v) - r(u)$ for any edge $e: u \to v$.

In order to define the effect of the slowdown and the retiming transformations we need to introduce two sets of operators on time functions, parametrized by positive integers k:

(1) *delay operators* θ^k such that $(\theta^k f)(t) = F(t - k)$ for any t;

(2) *spread operators* Ω^k such that $(\Omega^k f)(kt) = f(t)$ for any t, and undefined otherwise.

Theorem 2.2. *Let $S = \langle V, E, \delta, D, \pi\rangle$ be a semisystolic system. Let $\Phi_I = \{f_v \mid v \in V_I\}$ be a set of time functions associated with input nodes of S. Let $\Phi = \{f_v \mid v \in V\}$ be the least fix-point semantics of S with respect to Φ_I.*

(1) If $k > 0$, then $\Phi^{(k)} = \{\Omega^k f_v \mid v \in V\}$ is the least fix-point semantics of the k-slowdown semisystolic system $S^{(k)}$ with respect to the input time functions $\Omega^k f_v$ for $v \in V_I$.

(2) *If r is a retiming transformation of S, then $\Phi_r = \{\theta^{r(v)} f_v \mid v \in V\}$ is the least fix-point semantics of the retimed semisystolic system S_r, with respect to the input time functions $\theta^{r(v)} f_v$ for $v \in V_1$.*

It is often easier to design a semisystolic system for solving a problem than a systolic one. The main reason for that is that in semisystolic systems one can use broadcasting to transmit data without any delay, wherever they are needed. Because of that, semisystolic systems are often more transparent. On the other hand, the existence of delay-free interconnections requires an increase in the length of the clock cycle. It is therefore very desirable to be able to transform semisystolic systems into systolic ones with the same interconnection structure. Necessary and sufficient conditions for the existence of retiming transformations capable of doing that are well known [71]. The removing of broadcasting is the main use of retiming transformations. For some semisystolic systems S there is no retiming transformation to obtain a systolic system from S, but if first a proper slowdown transformation is applied to S, then the resulting semisystolic system can be retimed to obtain a systolic one. This is the main use of slowdown transformations. In this way quite a few well known and tricky systolic systems can be easily obtained from very natural networks.

Example 2.3. Let $A = \{a_{ij}\}$ be an $n \times n$ matrix such that its LU-decomposition into a lower triangular matrix $L = \{l_{ij}\}$ with 1s in the main diagonal, and into an upper diagonal matrix $U = \{u_{ij}\}$ can be computed by Gaussian elimination without pivoting. The elements of matrices L and U can be computed according to the following recurrences:

$$a_{ij}^0 = a_{ij}, \qquad a_{ij}^k = a_{ij}^{k-1} - l_{ik} u_{kj}$$

$$l_{ik} = \frac{a_{ik}^{k-1}}{a_{kk}^{k-1}}, \qquad u_{kj} = a_{kj}^{k-1}, \quad k = 1, 2, \ldots, n, \; k \leq i \leq n, k \leq j \leq n.$$

It is now easy to see that LU-decomposition of A, for the case $n = 4$, can be computed by the semisystolic system and the flow of data as shown in Fig. 2 [62]. Computations

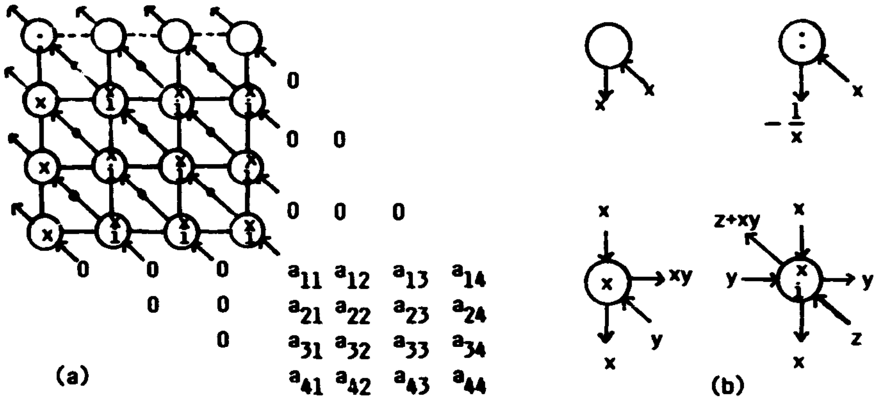

Fig. 2.

performed by four types of processors are shown in Fig. 2(b). Each diagonal has one register, vertical and horizontal edges have none.

There is no way to retime this semisystolic system to obtain a systolic one. On the other hand if at first the 3-slowdown transformation is applied, then it is easy to retime the resulting system in such a way that each edge has exactly one register. After some cosmetic changes one then quite easily obtains the well known systolic system of Kung and Leiserson [54] with quite tricky flow of computation.

Retiming transformations can also be used to optimize synchronous networks with respect to various cost criteria. For example, to minimize the clock cycle for the case when time needed to perform processing in each node is given, or to minimize the total number of registers. These optimization problems lead to various linear programming problems for which efficient algorithms are known [71]. Retiming transformations can also be used to deal with various problems concerning two level pipelining and fault-tolerance [52, 53].

Based on the model of semisystolic systems discussed in this section [70], the concept of a systolic flowchart scheme has been introduced in [2] to study syntactical properties of systolic systems. Equational axiomatization of such systolic flowchart schemes has been presented in [2]. Systolic flowchart schemes and their interpretations have been considered to be an algebraic framework useful for the study of systolic systems [3].

In [39], quite a different type of transformations of systolic networks is considered. They preserve timing of systolic computation flows but they may change topology of the underlying network. These transformations can also be used as a quite powerful systolic system design methodology.

3. Systolic system design methodologies

Transformations presented in the previous section represent powerful systolic system design tools. On a different level of abstraction, several, though quite restricted, design methodologies can be abstracted from the proofs of theorems dealing with the power of various classes of systolic automata (see Sections 5 and 6). These results allow, for example, the automatic design of a systolic trellis automaton to recognize any language being an intersection of a finite set of linear languages, directly from linear grammars that generate these languages.

Starting with [88] a variety of more or less formal techniques for systematic design of systolic systems has appeared [5, 15, 23, 28, 42, 69, 72-75, 77, 80]. Three of them are discussed now in more detail.

Ibarra and his co-workers have developed several characterizations of networks (of various architectures) of finite automata in terms of variants of sequential Turing machines. These characterizations are a base for a powerful design methodology.

One characterization is in terms of so-called full scan Turing machines (STM). There are actually several such machines, each for a different class of finite automata networks. They differ in the initial configurations, in positions where they read new input symbols, and how they react to endmarkers [49, 37]. We shall consider here only two of them, STM^d and STM^v. A STM M is a one-tape, one-head Turing machine with the external input (Fig. 3) to receive input words $a_1 \dots a_n\$$, where $a_i \in \Sigma$, and \$ is not in the input alphabet Σ. M begins a computation in the initial state q_0, and keeps performing right-to-left and left-to-right sweeps. An input is accepted if M writes an accepting symbol (from a $\Gamma_0 \subseteq \Gamma$, the tape alphabet). Figure 3b shows the initial configurations for STM^d and STM^v and complete sweeps. "R" shows where the reading of the external input takes place. "w" above a square emphasizes that some writing into that square occurs and "w(\$)" above a square indicates that "\$" is written into that square as soon as it is read from the external input. During their left-to-right sweeps STM always stay in a special state q_0, do not change tape contents and keep moving right until they meet \$ or the blank symbol λ. During right-to-left sweeps STM may change tape contents and states but they enter the state q_0 if and only if they come to the square with \$ or λ, then always a left-to-right sweep starts. The *sweep complexity* of M on an input is the least number of complete sweeps needed to accept the input.

The following theorem [49, 50] relates STM^d and STM^v with systolic trellis automata with diagonal and vertical acceptance [37] (Fig. 4). These automata have

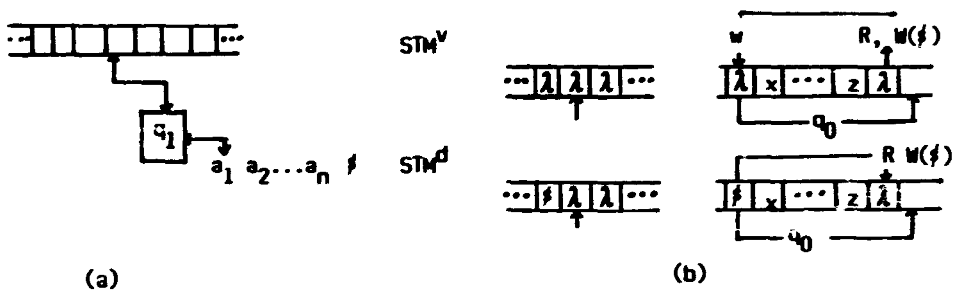

Fig. 3.

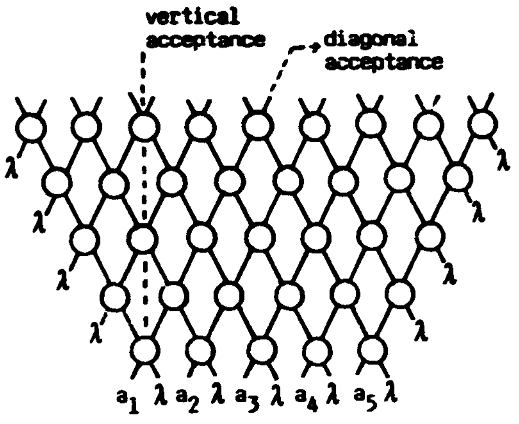
vertical acceptance
diagonal acceptance
λ
$a_1\ \lambda\ a_2\ \lambda\ a_3\ \lambda\ a_4\ \lambda\ a_5\ \lambda$

Fig. 4.

the form of an infinite network of identical memoryless processors, with the transition function g such that $g(\lambda, \lambda) = \lambda$. They actually represent time-space transformation of linear array computations, and therefore the following theorem can easily be reformulated in such terms.

Theorem 3.1. (1) *A language L is accepted by a systolic trellis automaton with vertical acceptance in time* $2T(n) - 1$, *if and only if it is accepted by a* STM^v *with the sweep complexity* $T(n)$.

(2) *A language L is accepted by a systolic trellis automaton with diagonal acceptance in time* $T(n)$, *if and only if L is accepted by a* STM^d *with the sweep complexity* $T(n)$.

While the previous theorem characterizes systolic automata (or linear arrays) as acceptors, the following one characterizes one-way two-dimensional cellular automata (O2DCA) as transducers (Fig. 5) in terms of so-called two-dimensional sequential machines (2DSM) (Fig. 6) [43, 48].

A 2DSM M with the input of n symbols operates on a two-dimensional tape of $n \times n$ squares. Initially all squares contain the symbol λ. Similarly as for STM, 2DSM also operate in sweeps. A sweep begins with M in a distinguished state q_0 and with the head on the leftmost square of the topmost row. M then reads an input symbol and moves through squares of the first row, from left to right, rewriting symbols and changing states (except into q_0) as a Turing machine does. After the rightmost square is visited, the head is reset to the first square of the next row in

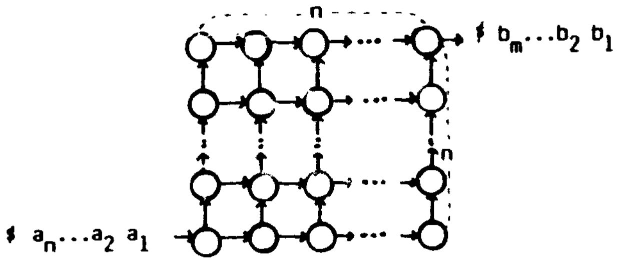

Fig. 5.

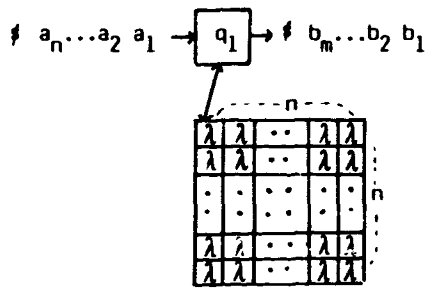

Fig. 6.

the state q_0. This action is repeated for all rows. At each step, new symbol and new state depend on the previous state, on the old symbol in the square being scanned and, if there is any, also on the symbol stored in the square just above the scanned square. After scanning the last square of the last row an output symbol is produced, and M is reset to the first square of the first row to start the next sweep. M is said to have *sweep complexity* $S(n)$ on an input $a_1 \dots a_n\$$ if it outputs \$ after at most $S(n)$ sweeps.

Theorem 3.2 (Ibarra [43]). *Let* $S(n) \geq n+1$. *A* 2DSM *with sweep complexity* $S(n)$ *can be simulated by a* O2DCA *in time* $S(n)+2n-2$, *and conversely.*

There are many other characterizations of array computations in terms of sequential machines. They have been used to derive new and efficient systolic systems for those tasks for which no such systolic systems are yet known. (No other design methodology has been so successful.) For example, the last theorem has been used to show that recognition and parsing of context-free languages can be done on O2DCA in linear time [16]. These characterizations have also been used to obtain new theoretical results. For example a generalization of Theorem 3.2 to higher dimension has been used to show [43] that $(k+1)$-head nondeterministic finite automata can be simulated by a O2DCA in time $(k+1)n+k-1$.

The second important class of systolic system design methodologies consists of various data dependence graph manipulating strategies. The main idea is to analyse a data dependence graph and then to transform it into the equivalent one that satisfies certain constraints that are natural abstractions of systolic and VLSI requirements. Methodologies of this type [36, 77, 79, 80, 83] have been especially successful for the design of systolic systems for matrix computations because in such cases one can naturally associate operations of computations with integer points of three- or four-dimensional space (see Fig. 7 for the data dependence graph for multiplication of matrices of degree 2, i.e. $c_{ij} = \Sigma_{k=1}^{2} a_{ik}b_{kj}$).

In order to obtain more easy manipulable data dependence graphs, with only local interconnections, a slight modification of basic algorithms is usually useful—this results mainly in the introduction of some forms of pipelining of input data.

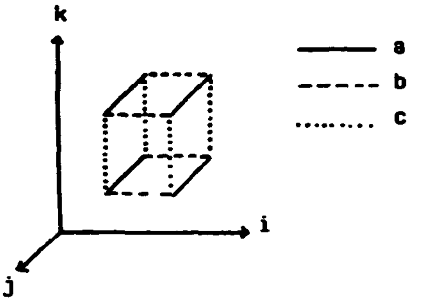

Fig. 7.

For matrix multiplication it has, for example, the form:

$$c(i,j,k) = c(i,j,k-1) + a(i,j,k-1)b(i-1,j,k),$$

$$a(i,j,k) = a(i,j-1,k),$$

$$b(i,j,k) = b(i-1,j,k).$$

Various geometric transformations, e.g. affine transformations of data dependence graphs [23], followed by their projection into a plane or a line, result in a variety of two- or one-dimensional systems.

The main problem is how to find, in a sufficiently systematic way, suitable transformations and projections. There have been attempts to solve this problem for computation tasks specified by some restricted specification languages. Perhaps the best known is the method developed by Quinton [77]. It can be applied to the design of systolic systems for computations that can be expressed as a set of uniform reccurent equations

$$\begin{aligned} U_1(z) &= f(U_1(z-\theta_1), \dots, U_p(z-\theta_p)) \\ U_2(z) &= U_2(z-\theta_2) \\ &\vdots \\ U_p(z) &= U_p(z-\theta_p) \end{aligned} \qquad (1)$$

over a convex set D of integer coordinates of the n-dimensional space. f is a p-nary function, and $\theta_1, \dots, \theta_p$ are "dependence vectors" from Z^n.

Quinton's method consists of two steps:

(i) to find a *timing function* $t: D \to N$, i.e. a schedule of computation that is compatible with dependences resulting from the equations (1);

(ii) to find an *allocation function* $a: Z^n \to Z^2$ that maps D into a finite set of integer points, that represent positions of processors of a systolic system, in such a way that concurrent computations are mapped into different processors and resulting interconnections of processors, as well as data flows, are sufficiently regular.

In [77] necessary and sufficient conditions are given for a *quasi-affine timing function*

$$t(z) = \lfloor \pi^T z - \alpha \rfloor, \quad \pi \in Q^n, \alpha \in Q \quad \text{(set of rationals)} \qquad (2)$$

to satisfy all above mentioned requirements. These conditions allow π and α to be chosen.

Once t is fixed, the task is to find an allocation function a such that $a(D)$ is finite, and $a(x) = a(y) \to t(x) \neq t(y)$ if x, y are in D. In [77] it is shown how to find a quasilinear allocation function. This function actually represents a projection of D along a properly chosen vector—a ray for D.

There have been many modifications and generalizations of this approach [36, 79, 80, 83]. Interactive software systems for the design of systolic system have also been built on this base (system DIASTOL in [28] and system S^4 in [83].) Of special interest is the approach developed by Sedukhin [80-83]. His goal has been to develop a methodology for finding all (in a reasonable sense) systolic systems for a given computational problem specified in some more general specification language. It is the language of *recurrence equations with linear dependences*

$$g(p)=f(g(p-\theta_1), g(p-\theta_2), \dots, g(p-\theta_k)) \tag{3}$$

where again p and $\theta_1, \dots, \theta_k$ are vectors from Z^n and

$$p-\theta_j = A_j p + b_j \quad \text{for } j=1,\dots,k,$$

where A_j are constant $n\times n$ matrices and b_j are vectors from Z^n. In the case that the rank of the matrix is $n-1$, which is the case for practically all known examples, then there is a method [80] to transform (3) into the pipelined form (1) with constant dependence vectors.

The vector π and the constant α from the timing function in (2) can also be obtained by solving an appropriate number of equations

$$t(z_j) = |\pi^T z_j - \alpha|$$

where $t(z_j)$, for points z_j, can be determined from the dependence graph by the maximal distance of the point z_j from the node representing the start of the computation process. After the timing function is specified, the next step is to project the data dependency graph along all directions that are not parallel to the hyperplane defined by the timing function and thereby all possible systolic architectures are obtained. For that it is of course important that projections preserve the nearest neighbour property and for that it is sufficient to consider as projection vectors μ, only vectors with coordinates -1, 0, 1. After excluding the null and symmetric directions of projection vectors, then the number of projections, and therefore of potentially different systolic systems, is $(3^n-1)/2$ which gives 13 for the most common case $n=3$ and 4 for $n=2$. The projections of nodes of the data dependence graph have to be conflict-free, with respect to the timing function, but this is achieved if vectors μ are not parallel to the timing hyperplane, i.e. if the scalar product (π, μ) is not zero. In [80-83], all systolic systems for matrix multiplication (13), LU-decomposition (13), the algebraic path problem (9), and discrete Fourier transformation (4) are derived, including some not previously known. Software system S^4 (System of Systolic Structure Synthesis) (see [83]) generates the set of possible transformations at each step of the systolic system synthesis processes and also provides tools for selecting "the best" systolic system.

Systolic system verification techniques are also an important part of systolic system design methodologies. One very natural idea [55] is to make use of the regularity of network interconnections and the regularity of data and computation flow. In many cases both the position of processors and the positions of data in data streams can, at any time moment, be naturally represented by integer points in the two-dimensional plane. This allows expression of the relation between the data and their

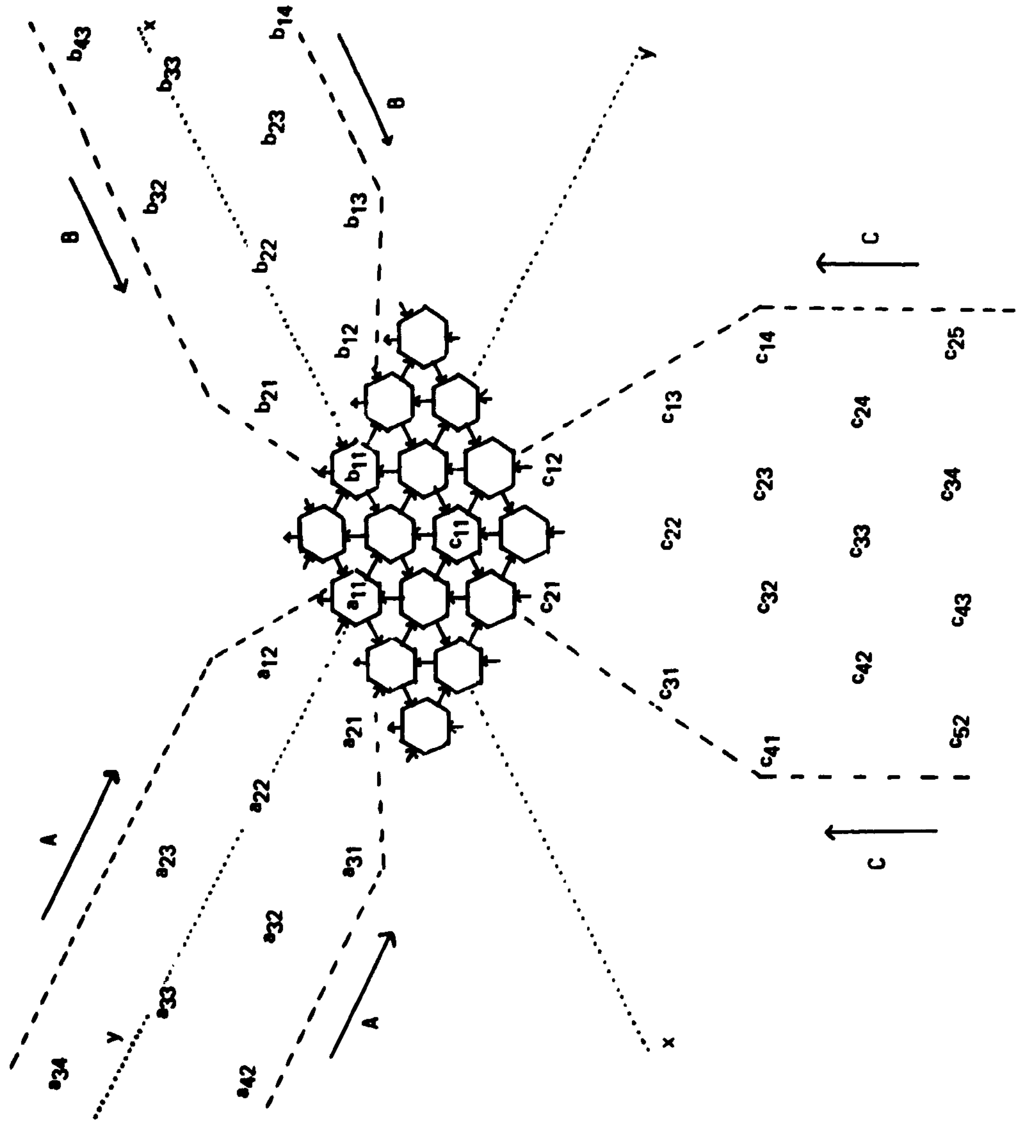
B
b43
x
b33
b32
b14
b23
b22
b13
B
b12
b21
b11
y
c11
a11
C
c14
c25
c13
c24
c12
c23
c34
c22
c33
c21
c32
c43
c31
c42
c41
c52
C
a12
a21
a22
A
a23
a31
a32
a33
y
a34
a42
A
x

Fig. 8.

positions at different time-moments by so-called *space-time-date equations* and these equations can then be used to show that correct data arrive at the processors at the correct time and on this basis the correctness of the *whole* systolic system is shown.

Example. The well-known systolic system [54] for multiplication of matrices of an arbitrary degree but with bandwidth 4 is shown in Fig. 8. If the proper coordinate system is chosen (represented by the dotted lines in Fig. 8), then space-time-data equations for the movement of elements a_{ij} of the matrix A, which relate the indices i, j and the coordinates (x, y) of position a_{ij} in time t, have the form

$$x-j-i=0,$$

$$y-i-2j+2-t=0.$$

Similar space-time-data equations can easily be derived for the movement of the elements of the matrices B and C. Using these equations one can show that whenever a c_{ij} arrives at a processor, then it meets there the proper elements of matrices A and B to make the computation that is needed.

More formal and semantics theory based development of systolic system design and verification framework is given in [38, 67, 78]. In [67, 89], it is shown how to develop and prove correctness of some systolic systems in the framework of *algebra of communicating processes*. In [78] *trace theory* framework is used to discuss and develop systolic systems in terms of their input/output behaviour.

4. Simulations

Simulation problems of three types are of great importance for the design of parallel networks.

(1) How to simulate efficiently networks of one parallel architecture on networks of another parallel architecture. It often happens that it is easier to design an algorithm for implementation on a particular architecture (e.g. on two-directional cellular rings), than on a slightly different architecture (on one-directional cellular rings), networks of which are physically easier to implement. Therefore any technique that shows how to simulate systematically and efficiently, networks of one parallel architecture on another architecture, represents an important network design tool.

(2) How to simulate efficiently large networks on smaller networks of the same parallel architecture. If one has to design a network for solving a particular problem, then it is usually very convenient to choose a network of the size that just matches the size of a given problem. This requires arbitrarily large networks to be considered. On the other hand, the size of available multiprocessor networks is, in practice, either fixed or with a severe upper bound.

(3) How to simulate, time and/or space efficiently, networks of a given parallel architecture on various models of sequential machines.

Quite a general concept of *simulation* (of one network on another) has been defined in [9]. It describes the case where one processor of a network N_1 may be simulated at different time moments by different processors of a network N_2 and, moreover, that an edge connection of N_1 is simulated by the whole path in N_2. In many cases it is, however, sufficient to use two simpler concepts of simulation: partial emulation and emulation.

Informally, a network N_2 *partially emulates* a network N_1, if to each processor P of N_1 a processor P' in N_2 can be associated in such a way that any computation on P in N_1 is simulated by a computation on P' in N_2. Similarly, N_2 *emulates* N_1, if to each processor P in N_1 a processor $\mu(P)$ in N_2 can be associated in such a way that for each edge $e: P_1 \to P_2$ in N_1, any communication along e is simulated by a communication along an edge from $\mu(P_1)$ to $\mu(P_2)$.

An emulation of N_1 on N_2 is called *computationally uniform* if the same number of processors of N_1 are mapped into each processor of N_2, and also the same number of edges of N_1 are mapped into each edge of N_2.

The concepts of *unrolling* and of the *isomorphism of unrollings* are important to establish simulation results. Informally, the unrolling of a network N with a set of nodes V, is the time-space transformation of the computational process or, in other words, an infinite data dependence graph with nodes (v, t), where $v \in V$ and t represents time. Isomorphism of unrollings is then the usual graph isomorphism.

Example (*Culik and Fris* [8]). Simulation between homogeneous networks on two-directional cellular rings (CR_n) (Fig. 9a) and homogeneous one-directional cellular rings (OCR_n) of n processors. Emulation of OCR_n on CR_n is trivial. In order to obtain a simulation in the opposite direction we proceed as follows: Let C_n be the network with n nodes, where each node v_j, $j = 1, 2, \ldots, n$ is connected with nodes v_j, $v_{j\oplus 1}$, $v_{j\oplus 2}$ by edges with the delay 1 (where $\oplus$ means the addition modulo n) (see Fig. 9b for C_9). The unrollings of CR_n and C_n are isomorphic; the isomorphism is established by the mapping $\delta:(v_i, t) \to (v_{j+t}, t)$. This implies that each homogeneous network on CR_n is simulated in real time on C_n and vice versa.

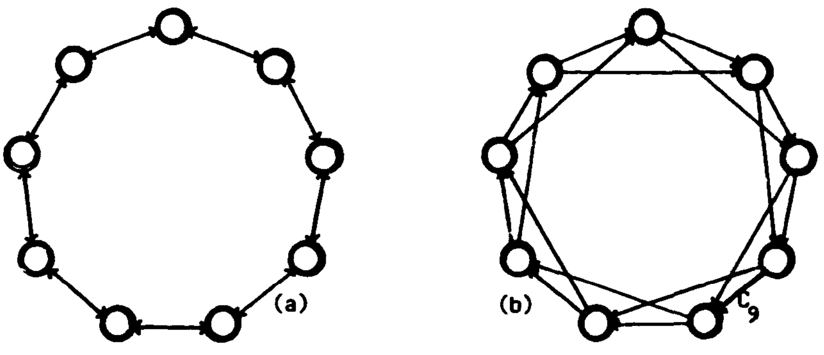

Fig. 9.

Moreover, C_n can clearly be partially emulated on OCR_n in such a way that some edges of C_n are simulated by paths of length 2 on OCR_n.

Similar simulations have been established between two-directional cellular arrays, one-directional cellular toroids and two-directional cellular toroids [8].

In connection with the study of complex systems the concept of totalistic CA has been introduced [91, 92] and investigated in various papers. A totalistic CA is a CA states of which are integers and a new state of a processor depends only on the sum of the old states of the processor and of its neighbours. It has been shown in [1] that for each CA C there exists a totalistic CA C' which simulates C without loss of time. This result has been generalized in [21] for cellular automata over arbitrary graphs.

Another interesting problem is to determine all possible computationally uniform emulations of large networks on smaller ones. It has been shown [4] that the number of computationally uniform emulations of CR_n on $\mathrm{CR}_{n/2}$ (as well as of two-dimensional cellular toroids of the size $n \times n$ on toroids of the size $\frac{1}{2}n \times \frac{1}{2}n$) is exponential (at least exponential). On the other hand, there are exactly six computationally uniform emulations of perfect shuffle networks of 2^n nodes on networks of 2^{n-1} nodes.

Another important problem is to find, for important classes of networks, say C, another class of networks, say C', such that on any network from C', every network from C can be emulated in a computationally uniform, or almost uniform, way. If such a class C' exists, then a fixed-size multiprocessor system with a network from C' can be used to emulate almost uniformly any multiprocessor system with a network from C; it is only necessary to sufficiently enlarge the memory of the processors and the width of the interconnections.

For rectangular arrays, such a class of networks, so-called *polymorphic arrays*, has been shown in [35]. They are borderless networks B_n of arrays of the size $S_n = F_n \times L_n$ where

$$F_1 = 1, \qquad F_2 = 1, \qquad F_j = F_{j-1} + F_{j-2} \quad \text{for } j > 2$$

are Fibonacci numbers and

$$L_1 = 1, \qquad L_2 = 2, \qquad L_j = L_{j-1} + L_{j-2} \quad \text{for } j > 2$$

are Lucas numbers, and a processor of B_n in the position (i, j) is connected with processors in the nodes

$$((i+1) \bmod F_n, (j+1) \bmod L_n), \quad ((i-1) \bmod F_n, (j-1) \bmod L_n),$$
$$((i+1) \bmod F_n, (j-1) \bmod L_n), \quad ((i-1) \bmod F_n, (j+1) \bmod L_n).$$

It has been shown in [35], that on any such network B_n, any rectangular array C with at most $S_n/\sqrt{5}$ processors, can be emulated in such a way that into each processor of B_n, at most one processor of C is mapped, and , moreover, arbitrarily large rectangular array C' can be emulated by B_n in such a way that the number of processors of C' mapped into one processor of B_n differs at most by $\mathrm{O}(\log S_n)$.

The last issue we deal with in this section concerns simulations of parallel networks on sequential machines. The following theorems summarize some results concerning simulations of arrays, trellises and tree networks on Turing machines and RAMs.

Theorem 4.1 (Chang et al. [17]). (1) *Any language accepted by a one-directional cellular tree network can be accepted by a deterministic Turing machine in space* $(\log^2 n)/(\log\log n)$.

(2) *Any language accepted by a one-directional cellular trellis network can be accepted by a deterministic Turing machine in space* $n\sqrt{n}$.

(3) *Any language accepted by a k-dimensional cellular array (of* n^k *processors) can be accepted by a deterministic Turing machine in space* $n^{2-1/k}$.

Theorem 4.2 (Ibarra [43], Černý and Gruska [10]). (1) *Any language accepted by a two-way k-dimensional cellular array can be accepted by a* RAM *in time* $O(n^{k+1}/\log n^{(k+1)/k})$.

(2) *Any language accepted by regular {modular} [regular and modular] (homogeneous) real-time trellis automaton can be accepted by a one-tape Turing machine in time* $O(n^3)$, $\{O(n^2\log n)\}$, $[O(n^2\sqrt{\log n})]$, $(O(n^2))$.

(Regular and modular trellis automata are defined in Section 6.)

5. Power of tree computations

One of the main goals in parallel computations is to do as much as possible in (poly)logarithmic time From this point of view tree networks of finite automata are perhaps the very basic model to investigate.

Two basic types of networks of finite automata have been intensively investigated. In both cases the underlying interconnection structure is an infinite leafless tree that is regular and nondegenerate in some reasonable way. In the case of *iterative tree automata* (ITA) [24], sequential input (output) goes to (from) the root processor (Fig. 1c) and only nonhomogeneous networks are of interest. We shall consider here only the case where the underlying tree is balanced (i.e. each node has the same number of sons), and for all relevant i, ith sons of all nodes are identical. In the case of *systolic tree automata* (STA), [94] the input is parallel (Fig. 1c), one input symbol per processor, and to the leftmost processors of the first level of processors with enough processors. Processors are *memoryless*, flow of computation is one-directional to the root, and most of the research has concentrated on the study of STA as acceptors. Regularity condition from [94] requires that there are only finitely many nonisomorphic subtrees and nondegenerativity condition requires that there is a constant $a > 1$ such that the jth level has at least a^j nodes.

The following theorem [19, 24, 25] shows that ITA are very powerful. In this theorem, by ITA(k) we denote ITA over a k-nary balanced tree and by ITM a modification of ITA with Turing machines instead of finite automata as processors.

Theorem 5.1. (1) *The family of languages accepted by* ITA *in time* $T(n)$ *is the same as the family of languages accepted by* ITM *in time* $T(n)$.

(2) *Any language accepted by a nondeterministic Turing machine in time* $T(n)$ *can be accepted by a* ITA(2) *in time* $O(T(n))$.

(3) *The family of languages accepted by linear time and real time* ITA *are identical.*

(4) *If* $2 \leq s < t$, *then the family of languages accepted by* ITA(s) *in linear time (in real time) is identical (is smaller) than the family of languages accepted in linear time (in real time) by* ITA(t).

(5) *The family of languages accepted in linear time by* ITA *contains all* CFL, *and it is closed under Boolean operations, concatenation, Kleene closure and morphism.*

In the above-mentioned model of ITA, and also of ITM, no restriction has been made concerning the depth of the trees really involved in particular computations, and therefore actually exponentially many processors can be active during a computation. It is therefore of importance to investigate how much can be done within depth-bounded ITA computations, i.e. computations on ITA where processors only of a restricted distance from the root can be used. Main results from [19] are summarized in the following theorem where $D(f(n))$-bounded ITA(k) denote ITA(k), the depth of computation of which is bounded by $f(n)$ for inputs of length n.

Theorem 5.2. (1) *A* $D(T(n))$*-bounded* ITM *with each processor being an* $S(n)$*-space bounded Turing machine, can be simulated by an* $D(O(T(n))+\log S(n))$*-bounded* ITA.

(2) $S(n)$*-space bounded on-line Turing machines are equivalent to* $D(\log S(n))$*-bounded* ITA.

(3) *Every* CFL *can be accepted by a* $D(O(\log n))$*-bounded* ITA. *Every deterministic* CFL *can be accepted in linear time by an* ITA.

Proof of the main results of the previous theorem is based on a clever simulation of pushdown stacks of size $S(n)$ on $D(\log S(n))$-bounded ITA. A similar idea is used in [6] to implement such data structures as stacks, queues, priority queues, deques, and dictionaries on $D(\log n)$-bounded ITA in such a way that, except for dictionary, all data structures have a unit response time. For dictionary operations the response time is $O(\log n)$ but instructions can be pipelined to the root at constant speed.

In the case of systolic tree automata [31, 94], attention has been paid so far to automata over trees with a finite base. (It is defined as a finite tree, all leaves of which have the same distance from the root (Fig. 10a–c).) An infinite leafless tree T is said to be over the finite base b, in short $T(b)$-tree, if it can be obtained from b by an infinite process at each step of which all leaves of the tree designed at the previous step are replaced by b-trees (see Fig. 10a1–c1) for trees with bases from Fig. 10a–c, respectively).

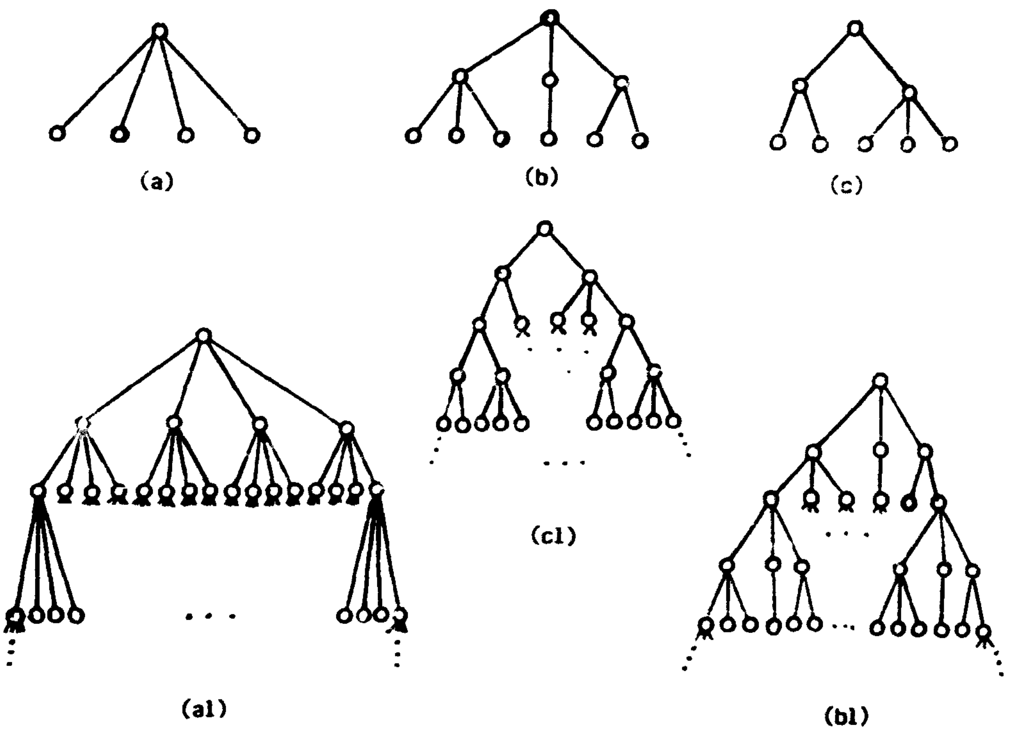

Fig. 10.

t-nary balanced STA, in short t-STA, are a special case of $T(b)$-trees (see Fig. 10a1). Let $T(b)$-STA denote the class of all systolic tree automata over $T(b)$ and $\mathcal{L}(T(b)\text{-STA})$, the family of languages accepted by $T(b)$-STA.

The following theorem summarizes relations between the recognition power of $T(b)$-STA for various bases [30].

Theorem 5.3. (1) *If s and t are (are not) powers of the same integer, then the families $\mathcal{L}(s\text{-STA})$ and $\mathcal{L}(t\text{-STA})$ are identical (are incomparable).*

(2) *If the base b has s leaves, then $\mathcal{L}(s\text{-STA}) \subseteq \mathcal{L}(T(b)\text{-STA})$, and the equality holds if and only if for any $0 \leq i < h(b)$ (where $h(b)$) is the depth of b), the set of prime divisors of the number of nodes of the* ith *level of b is a subset of prime divisors of s.*

The main results concerning the languages accepted by 2-STA are now presented.

Theorem 5.4. (1) *The family $\mathcal{L}(2\text{-STA})$ contains all regular languages, and also some languages very high in the language hierarchies. It is closed under Boolean operations, right concatenation with regular sets, restricted concatenation and selective concatenation. It is not closed under left concatenation with regular set, Kleene closure, morphism and ε-free morphism.*

(2) *Nondeterminstic 2-STA are as powerful as deterministic.*

(3) *The emptiness, finiteness and equivalence problems are decidable.*

Decidability of the emptiness problem for general STA is an open problem closely related to well-known decision problems from formal power series [11].

There is a characterization of balanced STA in terms of special Turing machines [46]. This characterization allows closure properties of $\mathcal{L}$(2-STA) to be proved using standard sequential computation techniques.

There are various modifications of STA concepts. Some of them consider different input modes. For example stable and superstable STA at which inputs can be submitted to processors of any level with sufficiently many processors and also to any chosen subsequence of processors at that level [11, 30]. STA where each input is first permuted (by a host) are considered in [40] and STA with an (infinite) program (represented by an infinite word, the initial part of which, of the same length as a given input word, is supplied to inputs of processors in parallel with the input word) are studied in [41]. Nonhomogeneous STA where each node-processor uniquely determines sons' processors are shown to be as powerful as homogeneous ones. STA where each node is also connected to the left brother of its left son and to the right brother of its right son, so called BC-trees, are investigated in [27, 33].

Fast recognition of regular languages by STA is of special interest. They can also be recognized by STA over *finite trees with a feedback* (which leads to an interesting recognition of regular languages by programmable systolic trees) [22]. Problems related to the optimization of STA as regular language recognizers are studied in [51]. STA have also been investigated as transducers [20] to obtain fast implementations of finite state automata realisable functions.

6. Power and structure of linear array computations

One-dimensional arrays of finite automata and their computations have been intensively investigated in the last years and the results show that they are not only a very basic model of parallel architecture but also a model with many interesting properties and of broader importance for theory of computation.

It has also turned out that many practically important computational problems can be solved sufficiently fast on linear arrays of simple processors [63, 65].

One-dimensional cellular automata (CA) have also been used as a basic model to study general problems of complexity because they seem to capture, in a reasonable sense, essential features responsible for complex behaviour of sytems composed from simple elements. This is also closely connected with the approach considering cellular automata as an alternative model of the physical world [84]. The underlying goal is to extract from such a study some general features of such phenomena as the self-organising behaviour, chaos and so on.

The importance of CA for such fundamental investigations and for the key applications in computing makes it very desirable to obtain more insight into the power and structure of linear array computations and to develop models and concepts suitable for this purpose.

The power of cellular automata depends on the amount of computational resources available (time, space, number of processors used), on the type of interconnections, and on the type of input.

Two basic models investigated are shown in Fig. 11a, c. They are one-dimensional cellular automata (with parallel input) and one-dimensional iterative arrays (with serial input). Their one-directional versions (OCA and OIA) are shown in Fig. 11b, d. Perhaps the most interesting case is the one in which there is no additional memory, i.e. for the input of the length n only n finite automata are used. Such CA and IA are called linear and denoted LCA and LIA (Fig. 11e, f), and their one-directional versions are denoted OLCA and OLIA.

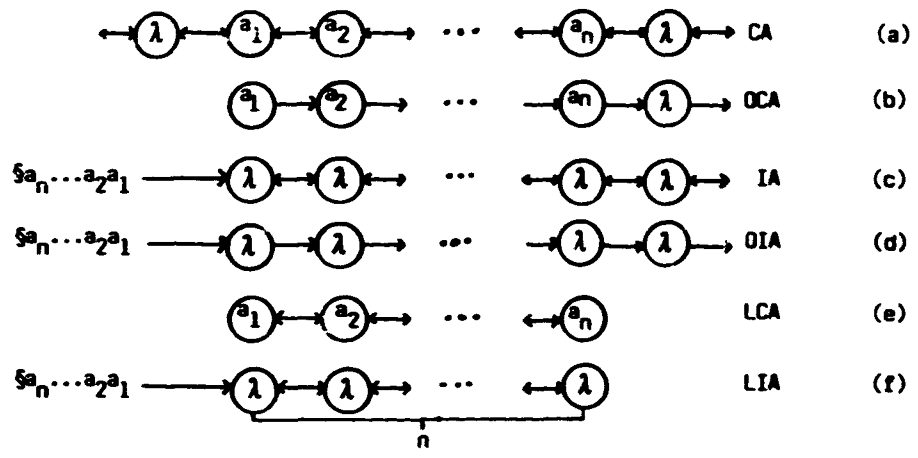

Fig. 11.

The basic problem is to determine how powerful is one-way communication (especially comparing with two-way communication) and what is the relation between computation in real-time (i.e. in time n for the input of the length n), pseudo-real time (in the time $2n$) and in linear time (in time cn for a constant c). The following theorem summarizes recent results that show surprising power of one-way communication and the relation of some of the open problems of this type to other well-known problems (for example, to some closure properties).

Theorem 6.1. (1) OLCA *accept exactly the same family of languages as* OLIA [45].

(2) *The family of languages accepted by* OLIA *is an* AFL, *it is closed under intersection, complementation and reversal*; *it contains some* PSPACE-*complete languages, all languages accepted*: *by linear-time bounded alternating* TM, *by* $\sqrt{n^c}$-*space bounded nondeterministic* TM *and by multihead two-way nondeterministic pushdown automata operating in* $c^{n/\log n}$ *time* (*and therefore all* CFL) [18].

(3) *If* $1 \le r < s$, *then the family of languages accepted by* OLIA *in time* n^r *is smaller than the family of languages accepted by* OLIA *in time* n^s [18].

(4) *The family of languages accepted by* OLCA *in linear-time is the same as the family of languages accepted by* OLIA *in pseudo-real-time, and the same as the family of languages accepted by* LCA *in real-time* [45].

(5) *The families of languages accepted by* LCA *in linear-time and by* LCA *in real-time are identical, if and only if the class of languages accepted by* LCA *in real-time is closed under reversal* [44].

(6) *If* LCA *are more powerful than* OLCA, *then nonlinear-time* LCA *are more powerful than linear-time* LCA [44].

The basic open problem is whether LCA are more powerful than OLCA.

A natural modification of the models of linear CA and IA is to consider models where the amount of processor used is $S(n)$ for an input of the length n. Various results for these models have been obtained in [18] also for the case $S(n) < n$.

A time space transformation of CA is shown in Fig. 12. By discarding vertical edges and adding processors on edge crossings we obtain the trellis network of memoryless processors shown in Fig. 4. In the case of the vertical acceptance in real-time we obtain so-called real-time trellis automata (RTTA) (see Fig. 13). The following theorem summarizes basic properties of the families of languages accepted by RTTA and their nondeterministic versions [13, 14, 46].

Theorem 6.2. (1) *The family of languages accepted by deterministic* RTTA *is the abstract family of deterministic languages which contains all linear context-free languages, some languages complete for polynomial time with respect to log-space*

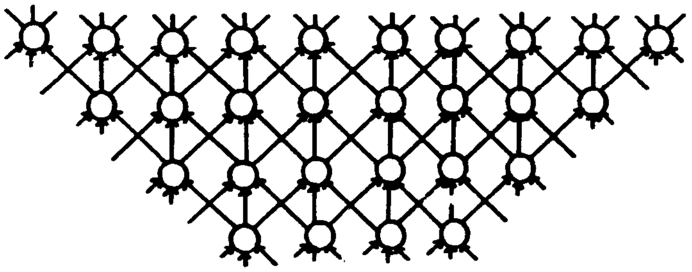

Fig. 12.

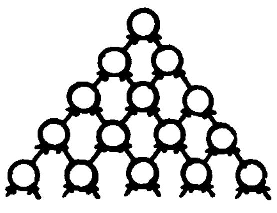

Fig. 13.

reducibility, and it is closed under Boolean operations, injective length multiplying morphism but not under morphism.

(2) *The family of languages accepted by nondeterministic* RTTA *is an* AFL *that contains all context-free languages, some* NP-*complete languages and it is contained in* DSPACE $(n \log n)$.

Real-time trellis automata are also a good model to study the power of various input modifications. For example, it has been shown that superstable RTTA are as powerful as ordinary ones [14]. Superstability requires that the acceptance does not depend either on the level (or row) of processors at which an input is submitted or on inputs of which processor symbols of an input word are submitted, provided the order is preserved. In [40], it is shown that the power of RTTA can be increased if input can be permuted (for example, by the host) before processing begins, and basic properties of languages accepted by RTTA with permuted inputs are investigated. The power of RTTA can also be increased if with any input of length n, the prefix of the length n of a fixed infinite word (called program) is also submitted in parallel to inputs of processors [41]. In [86], a special input mode for linear arrays, called pipeline processing, has been considered.

A trellis automaton can formally be defined as a quintuple $A = \langle \Sigma, \Gamma, \Gamma', \lambda, g \rangle$, where Σ is the input alphabet, Γ is the operating alphabet, $\lambda \in \Gamma$, $\Gamma' \subseteq \Gamma$, is the accepting alphabet, and $g : \Gamma \times \Gamma \to \Gamma$ is the transition function such that $g(\lambda, \lambda) = \lambda$. With the automaton A one can associate the algebra $\mathbf{E} = \langle \Gamma, l, r, * \rangle$ of signature (1, 1, 2), where $l(a) = g(\lambda, a)$ and $r(a) = g(a, \lambda)$ for $a \in \Gamma - \{\lambda\}$, and $a * b = g(a, b)$ for $a, b \in \Gamma - \{\lambda\}$. Similarly with any algebra of signature (1, 1, 2) one can associate a class of trellis automata that differ only in the input alphabet and in the accepting alphabet.

With any algebra $\mathbf{E} = \langle \Gamma, l, r, * \rangle$ of signature (1, 1, 2) and any word $w = w_0 \ldots w_n \in \Gamma^{n+1}$, one can associate the mapping GPT($\mathbf{E}$, w) that is defined on $\{(i, j) \mid i \in N, j \in N, i + j \geq n\}$ as follows:

$$\mathrm{GPT}(\mathbf{E}, w)(i, j) = w_i \quad \text{if } i + j = n,$$

$$\mathrm{GPT}(\mathbf{E}, w)(0, j) = l(\mathrm{GPT}(\mathbf{E}, w)(0, j-1)) \quad \text{if } j > n,$$

$$\mathrm{GPT}(\mathbf{E}, w)(i, 0) = r(\mathrm{GPT}(\mathbf{E}, w)(i-1, 0)) \quad \text{if } i > n,$$

$$\mathrm{GPT}(\mathbf{E}, w)(i, j) = (\mathrm{GPT}(\mathbf{E}, w)(i-1, j)) * (\mathrm{GPT}(\mathbf{E}, w)(i, j-1)) \quad \text{if } i + j > n, ij \neq 0.$$

The mapping GPT($\mathbf{E}$, w) is called the generalized Pascal triangle over (the algebra) $\mathbf{E}$ and (the word) w, and it can be depicted in the form shown in Fig. 14, where $w_{i,j} = \mathrm{GPT}(\mathbf{E}, w)(i, j)$. GPT($\mathbf{E}$, 1) for the algebra $\mathbf{E} = \langle N, \mathrm{id}, \mathrm{id}, + \rangle$ is the usual Pascal triangle, and for the algebra $\mathbf{E} = \langle \{0, 1\}, \mathrm{id}, \mathrm{id}, \oplus \rangle$, with $\oplus$ being the addition modulo 2, the GPT ($\mathbf{E}$, 1) is shown in Fig. 15.

$$\begin{array}{c}
w_{0,n}\ w_{1,n-1}\cdots w_{n-1,1}w_{n,0}\\
w_{0,n+1}w_{1,n}\cdots\cdots\cdots w_{n,1}\ w_{n+1,0}\\
w_{0,n+2}w_{1,n+1}\ldots\ldots\ldots w_{n+1,1}w_{n+2,0}\\
w_{0,n+3}w_{1,n+2}\cdots\cdots\cdots\cdots w_{n+2,1}w_{n+3,0}\\
\cdot\quad\cdot\qquad\qquad\qquad\qquad\qquad\qquad\cdot\quad\cdot\\
\cdot\quad\cdot\qquad\qquad\qquad\qquad\qquad\qquad\qquad\cdot\quad\cdot\\
\cdot\quad\cdot\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\cdot\quad\cdot
\end{array}$$

Fig. 14.

$$\begin{array}{c}
1\\
1\ 1\\
1\ 0\ 1\\
1\ 1\ 1\ 1\\
1\ 0\ 0\ 0\ 1\\
1\ 1\ 0\ 0\ 1\ 1\\
1\ 0\ 1\ 0\ 1\ 0\ 1\\
1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\\
1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1
\end{array}$$

Fig. 15.

Generalized Pascal triangles can also been seen as special two-dimensional infinite words that can be considered as a new abstract model of linear array computations. This new model naturally gives a rise to a new set of problems, for example:

(1) What is the structure of diagonals, columns and other components of GPT?

(2) What can be said about the density of occurrences of elements in GPT?

(3) When are the basic decision problems concerning the structure of GPT decidable? (For example the problem to determine whether a given symbol (a word or a pattern) occurs at least once (or infinitely often) in a given GPT.)

(4) What are the conditions for a complete GPT(E, w) to have the so-called self-embedding property (i.e. that there exist integers p, q such that GPT(E, w)($i+p$, $j+q$) = GPT(E, w)(i, j) for any i, j in N)?

(5) With any GPT one can associate the language of row-words. What are the properties of such languages?

All these problems are related to basic problems concerning linear array computations. In this way the model of generalized Pascal triangles brings new techniques into the study of linear array computations. It also relates structural and combinatorial properties of GPT (and thereby also of linear array computations) and algebraic properties of the underlying algebras.

Generalized Pascal triangles have recently been intensively investigated especially by Korec [59]. Some of the results are now presented.

In order to study properties of diagonals of GPT, it is convenient to consider instead of algebras $E = \langle A, l, r, * \rangle$ of the signature (1, 1, 2), the algebras $E' = \langle A, K, l, r, * \rangle$ of the signature (0, 1, 1, 2), and to consider GPT(E', K), in short GPT(E'), as (complete) GPT over E'.

Diagonals of GPT are clearly ultimately periodic. Let LPER(E', k) denote the smallest period of the kth left diagonal of the GPT(E'). Clearly LPER(E', k) $\leq |A|^k$. The algebra E' is said to be the *algebra with maximal left periods* if LPER(E', k) $= |A|^k$ for any $k \in N$.

Theorem 6.3 (Kochol [56]). (i) *$E' = \langle A, K, l, r, * \rangle$ is the algebra with maximal left periods, if and only if every word $u \in A^*$ is the prefix of at least one (of infinitely many) rows of* GPT(E').

(ii) *There is an algebra of the signature* (0, 1, 1, 2) *of cardinality n with maximal left periods, if and only if n is odd. There are exactly six different algebras of cardinality* 3 *with maximal left periods.*

(iii) *If $E' = \langle A, K, l, r, * \rangle$ is the algebra with maximal left periods, then the operation l is a cyclic permutation of A, the operation $*$ is not associative, and for every $a, b \in A$ there exists just one $x \in A$ such that $a * x = b$.*

The next problem we shall deal with is the density of occurrences of particular elements in GPT.

For an algebra $E = \langle A, l, r, * \rangle$, for $w \in A^*$ and $x \in A$, let us define the density of the occurrence of x in the GPT(E, w) as follows:

$$\text{DENS}(E, w, x) = \lim_{n \to \infty} \frac{\text{number of } x \text{ in first } n \text{ rows of GPT}(A, w)}{\text{number of all elements in first } n \text{ rows of GPT}(A, w)}.$$

The next theorem says that from the point of view of density, there exists, surprisingly, a "universal" algebra at which one can achieve any feasible density of elements by choosing a proper initial word.

Theorem 6.4 (Korec [60]). *For any $s \in N$ there exists a finite algebra $E = \langle A, k, r, * \rangle$ such that $\{1, 2, \ldots, s\} \subset A$, and for every ordered s-tuple $(\alpha_1, \ldots, \alpha_s)$ of nonnegative rationals such that*

$$\alpha_1 + \alpha_2 + \cdots + \alpha_s < 1$$

there is a $w \in A^*$ such that DENS(E, w, i) $= \alpha_i$.

Decision problems, the self-embedding problems, and row-language characterization problems are related, and they are also nicely related to algebraic properties of the underlying algebras as the following theorem shows [59, 60]. (The concept of simple semilinear language (of degree k) that is used in the following theorem is defined as follows. A language $L \subset \Sigma^*$ is called simple linear (of degree k) if it has the form $L = \{u_1 v_1^i u_2 v_2^i \ldots u_k v_k^i u_{k+1} \mid u_1, v_1, \ldots, u_k, v_k, u_{k+1}$ are fixed words in

Σ and $i \geq 0\}$, a language is called simple semilinear (of degree k) if it is a union of finitely many simple linear languages (of degree k).)

Theorem 6.5 (Korec [58, 59]). (i) *The problem whether a given element occurs at least once (infinitely often) [and many other related problems] is undecidable even for* GPT *over finite algebras with a commutative binary operation* $*$.

(ii) *All decision problems mentioned in* (i) *[and many other related problems] are decidable for those* GPT, *row-languages of which are simple semilinear.*

(iii) *If a* GPT *has the self-embedding property, then the corresponding row-language is simple similinear and context-free.*

(iv) *If the binary operation of the algebra* $\mathbf{E}$ *is idempotent and associative, then all* GPT *over* $\mathbf{E}$ *have the self-embedding property.*

Some GPT can be designed from simple modules (actually finite and rectangular two-dimensional words) using simple recurrences; for example GPT from Fig. 15 can also be obtained using reccurrences from Fig. 16. This has led to the investigation of so-called modular two-dimensional words, in short modular trellises, as mappings $T: N \times N \to \Gamma$ that can be designed in such a modular way. Formally, modular trellises can be defined in a way that is a two-dimensional generalization of the way a Thue–Morse infinite word is defined using iterated morphisms.

A trellis T over an alphabet Γ is said to be strictly (p, q)-modular if

$$T = \lim_{j \to \infty} \mu^j(a)$$

where μ maps Γ into rectangular two-dimensional words

$$\mu(b) = \begin{matrix} b_{11} \cdots b_{1p} \\ \vdots \quad \ddots \quad \vdots \\ b_{q1} \cdots b_{qp} \end{matrix}$$

such that $a_{11} = a$. A trellis T over the alphabet Γ is said to be (p, q)-modular if T can be obtained from a strictly (p, q)-modular trellis T' over an alphabet Γ' by renaming (i.e. if $T(i, j) = \theta(T'(i, j))$ where $\theta: \Gamma' \to \Gamma$ is a mapping). A trellis T is said to be [strictly] modular if it is [strictly] (p, q)-modular for some p, q. For example the trellis depicted in Fig. 14 is (2, 2)-strictly modular.

$$A_0 = 1 \qquad\qquad B_0 = 0$$

$$A_{j+1} = \begin{array}{ccc} & A_j & \\ A_j & & A_j \\ & B_j & \end{array} \qquad B_{j+1} = \begin{array}{ccc} & B_j & \\ B_j & & B_j \\ & B_j & \end{array}$$

Fig 16.

Modular and strictly modular trellises have been investigated in [9] where two quite different characterizations of them have also been derived. The first one characterizes (p, q)-modular trellises as exactly those trellises T for which $T(i, j)$ can be computed by a so-called (p, q)-sorting automaton. (It is a finite automaton such that if it receives on its input (in parallel) integers i and j (in p-nary and q-nary notation, respectively), then it comes to the state that uniquely determines $T(i, j)$). Due to this characterization one can determine $T(i, j)$ for a modular trellis in time $O(\log(i+j))$, while one seems to need, in general, time $\theta(i+j)$ to compute $T(i, j)$ for a GPT T. The second characterization characterizes modular trellises as exactly those trellises that are fix-points of special substitutions on trellises. (A substitution first partitions a trellis into rectangular words of the same size and then replaces each such two-dimensional subwords by another two-dimensional finite word, in general of a different size, but in all cases replacement is by subwords of the same size.) It has been shown that the family of modular trellises is quite robust; it is closed under various operations on trellises. Modularity of trellises of a special type has also been studied. For example it has been shown [61] that a Pascal triangle modulo p is modular (strictly modular), if and only if p is a prime power (a prime).

RTTA is also a suitable model to study the power of nonhomogeneity in systolic trellis computations. Three types of nonhomogeneous systolic RTTA have been investigated in more detail [10, 13, 46, 61]. Regular RTTA are such nonhomogeneous RTTA that their processor distribution forms a GPT. In an analogical way modular RTTA are defined as nonhomogeneous RTTA such that their processor distribution forms a modular trellis. It has been shown that in the deterministic case both regular and modular RTTA are more powerful than homogeneous ones. On the other hand nondeterministic regular RTTA are as powerful as nondeterministic homogeneous RTTA. (Whether the same holds true for modular RTTA is an open problem.) Regular and also modular RTTA may have a large number of different types of processors and quite complicated distribution of them. It is therefore interesting that for each regular (modular) RTTA one can design an equivalent regular (modular) RTTA that uses only processors of two types [10, 57]. This result also indicates that the concepts of regular and modular RTTA are a reasonable generalization of the concept of homogeneous RTTA.

In [61] the third type of nonhomogeneous RTTA has been investigated, semilinear RTTA. They are more powerful than homogeneous RTTA, and they seem to be less powerful than regular and modular RTTA. A nonhomogeneous RTTA is said to be semilinear of the degree k, if the set of the row words of the underlying trellis is semilinear of degree k.

It has been shown [61] that to every semilinear RTTA there is an equivalent semilinear RTTA that is both regular and modular. Moreover every semilinear RTTA is equivalent with a RTTA that is both regular and semilinear of degree 3 and with a semilinear RTTA of degree 2.

We have considered here systolic tree and trellis automata as two very basic models of parallel computations. A natural question is what is the relation between their power. It has been shown in [13] that families of languages accepted by systolic tree automata over balanced trees and by homogeneous RTTA are incomparable. On the other hand it has recently been shown [93] that each language accepted by a systolic tree automaton over a balanced tree can be accepted by a regular RTTA and that simulation of systolic tree automata over a balanced tree on regular RTTA can be done in quite a universal way.

References

[1] J. Albert and K. Culik II, Simple universal cellular automaton and its one-way and totalistic version, *Complex Systems* **1** (1987) 1-16.
[2] M. Bartha, An equational axiomatization of systolic systems, *Theoret. Comput. Sci.* **55** (1987) 265-289.
[3] M. Bartha, Interpretations of systolic flowchart schemes, TR, Bolyai Institute, Szeged, 1989.
[4] H. L. Bodlaender and J. van Leeuwen, Simulation of large networks on smaller networks, Dept. of Computer Science, TR-RUL-CS-84-4, University of Utrecht, 1984.
[5] S. D. Brooks, Reasoning about synchronous systems, Dept. of Computer Science, TR-CMU-CS-84-145, Carnegie-Mellon University, 1984.
[6] J. H. Chang, M. J. Chung, O. H. Ibarra and K. K. Rao, Systolic tree implementations of data structures, Dept. of Computer Science, TR-85-32, University of Minnesota, 1985.
[7] K. Culik II and C. Choffrut, On real-time cellular automata and trellis automata, *Acta Cybernetica* **21** (1984) 393-407.
[8] K. Culik II and I. Fris, Topological transformation as a tool in the design of systolic systems, *Theoret. Comput. Sci.* **37** (1985) 183-216.
[9] A. Černý and J. Gruska, Modular trellises, in: G. Rozenberg and A. Salomaa (eds.), *The Book of L* (Springer, Berlin, 1985) 45-61.
[10] A. Černý and J. Gruska, Modular real time trellis automata, *Fund. Inform.* **IX** (1986) 253-282.
[11] K. Culik II, J. Gruska and A. Salomaa, On a family of L languages resulting from systolic tree automata, *Theoret. Comput. Sci.* **23** (1983) 231-242.
[12] K. Culik II, J. Gruska and A. Salomaa, Systolic automata for VLSI on balanced trees. *Acta Inform.* **18** (1983) 335-344.
[13] K. Culik II, J. Gruska and A. Salomaa, Systolic trellis automata, Parts I and II, *Intern. J. Comput. Math.* **15** (1984) 195-212; **16** (1984) 3-22.
[14] K. Culik II, J. Gruska and A. Salomaa, Systolic trellis automata: stability, decidability and complexity, *Inform. and Control* **71** (1986) 218-230.
[15] M. Chen, Very-high level programming in Crystal, Dept. of Computer Science, TR 506, Yale University, 1986.
[16] J. H. Chang, O. H. Ibarra and M. A. Palis, Parallel parsing on a one-way array of finite state machines, Dept. of Computer Science TR-85-20, University of Minnesota, 1986.
[17] J. H. Chang, O. H. Ibarra and M. A. Palis, Efficient simulations of simple models of parallel computation by time-bounded ATM's and space bounded TM's, in: *Proc. ICALP'88*, Lecture Notes in Computer Science 317 (Springer, Berlin, 1988) 119-132.
[18] J. H. Chang, O. H. Ibarra and A. Vergis, On the power of one-way communications, Dept. of Computer Science, TR-86-11, University of Minnesota, 1986.
[19] K. Culik II, O. H. Ibarra and S. Yu, Iterative tree arrays with logarithmic depth, *Internat. J. Comput. Math.* **20** (1986) 187-204.
[20] K. Culik II, H. Jürgensen and K. Mak, Systolic tree architecture for some standard functions, Dept. of Computer Science, TR-140, University of Western Ontario, 1985.
[21] K. Culik II and J. Karhumäki, On totalistic systolic networks, *Inform. Process. Lett.* **26** (1988) 231-236.
[22] K. Culik II and H. Jürgensen, Programmable finite automata, for VLSI, *Internat. J. Comput. Math.* **14** (1983) 259-275.

[23] P. Cappello and K. Steiglitz, Unifying VLSI design with geometric transformations, in: *Proc. IEEE Internat. Conf. on Parallel Processing* (1988) 448-451.

[24] K. Culik II and S. Yu, Iterative tree automata, *Theoret. Comput. Sci.* **32** (1984) 227-247.

[25] K. Culik II and S. Yu, Real-time, pseudo real-time and linear time ITA, *Theoret. Comput. Sci.* **47** (1986) 15-26.

[26] D. de Baer and J. Paredaens, A formal definition for systolic systems, Dept. of Mathematics, TR, University of Antwerp, 1984.

[27] E. Fachini, R. Francese, M. Napoli and D. Parente, BC-tree systolic automata: characterization and property, *Comput. Artificial Intelligence* **8** (1989) 53-82.

[28] P. Frison, P. Gachet and P. Quinton, Designing systolic arrays with DIASTOL, RR-578, INRIA, 1986.

[29] A. L. Fischer and H. T. Kung, Synchronising large VLSI processor arrays, *IEEE Trans. Comput.* **34** (1983) 734-740.

[30] E. Fachini, A. Maggiolo Schettini, G. Resta and D. Sangiorigi, Nonacceptability criteria and closure properties for the class of languages accepted by binary systolic tree automata, Dept. Informatica ed Applicazioni, TR-54-88, University of Salerno, 1988.

[31] E. Fachini, A. Maggiolo Schettini, G. Resta and D. Sangiorgi, Some structural properties of systolic tree automata, Dept. Informatica ed Applicazioni, TR-55-88, University of Salerno, 1988.

[32] M. A. Frumkin, Systolic programming (in Russian), *Voprosy Kibernet.* (1988) 101-120.

[33] E. Fachini and M. Napoli, C-tree systolic automata, *Theoret. Comput. Sci.* **56** (1988) 155-186.

[34] A. L. Fischer, Memory and modularity in systolic array implementation, in: *Proc.* 1985 *Conf. on Parallel Processing* (1985) 99-101.

[35] A. Fiat and A. Shamir, Polymorphic arrays: a novel VLSI layout for systolic computers, in: *Proc. STOC* (1984) 37-45.

[36] N. Faroughi and M. A. Shanblatt, Systematic generation and enumeration of systolic arrays from the algorithms, in: *Proc. Internat. Conf. on Parallel Processing*, University Parc (1987) 844-847.

[37] J. Gruska, Systolic automata: power, characterization, nonhomogeneity, in: *Proc. MFCS'84*, Lecture Notes in Computer Science **176** (Springer, Berlin, 1984) 32-49.

[38] M. Hennessy, Proving systolic systems correct, in: *Proc. TOPLAS* (1986) 344-387.

[39] S. W. Hornick, A unified approach to the analysis and synthesis of systolic arrays, Dept. of Electrical Engineering, TR-1039, University of Illinois, 1985.

[40] J. Hromkovič and D. Pardubská, Some complexity aspects of VLSI computations. On the power of input bit permutations in tree and trellis automata, *Comput. Artificial Intelligence* **7** (1988) 397-412.

[41] J. Hromkovič and D. Pardubská, Some complexity aspects of VLSI computations, VLSI circuits with programs, *Comput. Artificial Intelligence* **7** (1988) 481-496.

[42] C. H. Huang and C. Lengauer, An implemented method for incremental systolic design, in: *Proc. PARLE*, Lecture Notes in Computer Science **259** (Springer, Berlin, 1987) 160-177.

[43] O. H. Ibarra, Systolic arrays: characterization and complexity, in: *Proc. MFCS*, Lecture Notes in Computer Science **233** (Springer, Berlin, 1986) 140-153.

[44] O. H. Ibarra and T. Jiang, On some open problems concerning cellular arrays, Dept. of Computer Science, TR, University of Minnesota, 1987.

[45] O. H. Ibarra and T. Jiang, On one-way cellular arrays, *SIAM J. Comput.* **16** (1987) 1135-1153.

[46] O. H. Ibarra and S. M. Kim, Characterizations and computational complexity of systolic trellis automata, *Theoret. Comput. Sci.* **29** (1984) 123-153.

[47] O. H. Ibarra and S. M. Kim, A characterization of systolic binary tree automata and applications, *Acta Inform.* **21** (1984) 193-207.

[48] O. H. Ibarra and M. A. Palis, Two-dimensional systolic arrays: characterizations and applications, *Theoret. Comput. Sci.* **57** (1988) 47-86.

[49] O. H. Ibarra, M. A. Palis and S. M. Kim, Designing systolic algorithms using sequential machines, in: *Proc. STOC* (1984) 46-55.

[50] H. O. Ibarra, M. A. Palis and S. M. Kim, Some results concerning linear iterative (systolic) arrays, *J. Parallel Distributive Comput.* **2** (1985) 182-218.

[51] H. Jürgensen and A. Salomaa, Syntactic monoids in the construction of systolic tree automata. *Internat. J. Comput. Inform. Sci.* **14** (1985) 35-49.

[52] H. T. Kung and M. S. Lam, Fault-tolerance and two-level pipelining in VLSI systolic arrays, in: *Proc. Conf. Advanced Research in VLSI*, MIT (1984).

[53] H. T. Kung and M. S. Lam, Wafer-scale integration and two level pipelined implementations of systolic arrays, *J. Parallel Distributed Process.* **1** (1984).

[54] H. T Kung and C. E. Leiserson, Systolic arrays (for VLSI), in: *Sparse Matrix Proceedings* (Soc. for industrial and Applied Mathematics, 1978) 256-282.

[55] C. J. Kuo, B. C. Levy and B. R. Musicus, The specification and verification of systolic wave algorithms, TR, Dept. of Electrical Engineering and Computer Sciences, MIT, Cambridge, 1984.

[56] M. Kochol, Generalized Pascal triangles with maximal left periods, *Comput. Artificial Intelligence* 6 (1987) 54-76.

[57] I. Korec, Two kinds of processors are sufficient and large operating alphabets are necessary for regular trellis automata languages, *Bull. EATCS* 23 (1984) 35-42.

[58] I. Korec, Generalized Pascal triangles, decidability results, *Acta Math. Univ. Comenian.* 46-47 (1985) 93-130.

[59] I. Korec, Generalized Pascal triangles (in Slovak), Doktoral Thesis, Comenius University.

[60] I. Korec, Asymptotical densities in genealized Pascal triangles, *Comput. Artificial Intelligence* 5 (1986) 187-198.

[61] I. Korec, Semilinear real-time trellis automata, in: *Proc. FCT'89*, Lecture Notes in Computer Science (Springer, Berlin, 1987).

[62] S. Y. Kung, On supercomputing with systolic/wavefront arary processors, *Proc. IEEE* 72 (1984) 867-884.

[63] H. T. Kung, Systolic algorithms for the CMU warp processor, Dept. of Computer Science, TR-CMU-CS-84-158, 1984.

[64] H. T. Kung, Memory requirements for balanced computer architectures, Dept. of Computer Science, TR, Carnegie-Mellon University, 1985.

[65] H. T. Kung, Warp Demo, Dept. of Computer Science, Carnegie-Mellon University, 1986.

[66] H. T. Kung, Special-purpose supercomputers, in: *Information Processing IFIP'86*. Participants Edition (1986) 565-570.

[67] L. Kossen, and W. P. Weijland, Correctness proofs of systolic algorithms: palindroms and sorting, Dept. of Computer Science, TR-FVI-87-04, University of Amsterdam, 1987.

[68] F. T. Leighton and C. E. Leiserson, Water scale integration of systolic arrays, *IEEE Trans. Comput.* 34 (1985) 448-461.

[69] M. Lam and J. Mostow, A transformational model of VLSI systolic design, in: *Proc. 6th Internat. Symp. on Computer Hardware Description Languages and their Applications, IFIP* (1983) 65-67.

[70] C. E. Leiserson and J. B. Saxe, Optimizing synchronous systems, in: *Proc. FOCS* (1981) 23-36.

[71] C. E. Leiserson, F. M. Rose and J. B. Saxe, Optimising synchronous circuitry by retiming, in: *Proc. Caltech Conf. on Very Large Scale Integration* (1983) 87-116.

[72] G.-J. Li and B. W. Wah, The design of optimal systolic arrays, *IEEE Trans. Comput.* 34 (1985) 66-77.

[73] B. Lisper, Description and synthesis of systolic arrays, Dept. of Numerical Analysis and Computing Science, The Royal Institute of Technology, Stockholm, 1986.

[74] D. I. Moldovan, On the design of algorithms for VLSI systems, *Proc. IEEE* 71 (1983).

[75] W. L. Miranker and A. Winkler, Space-time representations of computational structures, *Computing* 32 (1984) 93-114.

[76] D. Pardubská, Closure properties of the family of languages defined by systolic tree automata, *Comput. Artificial Intelligence* 7 (1988) 59-64.

[77] P. Quinton, The systematic design of systolic arrays, TR-193, IRISA, 1983.

[78] M. Rem, Trace theory and systolic computations, Dept. of Mathematics and Computer Science, TR, Eindhoven University of Technology, 1988.

[79] S. Rajopadhye, S. Purushothaman and R. Fujimoto, On synthesising systolic arrays from recurrent equations with linear dependencies, in: *Proc. Foundations of Software Technology and Theoretical Computer Science*, Lecture Notes in Computer Science 241 (Springer, Berlin, 1986) 488-503.

[80] S. G. Sedukhin, Systematic approach to the design of VLSI networks (in Russian), Preprint, Academy of Sciences, Novosibirsk, 1985.

[81] S. G. Sedukhin, Design and analysis of systolic algorithms for algebraic path problem, TR, Computing Center, Academy of Sciences, Novosibirsk, 1987.

[82] S. G. Sedukhin, Design and analysis of systolic algorithms for the algebraic path problems, TR, Computing Center, Academy of Sciences, Novosibirsk, 1988.

[83] S. G. Sedukhin and E. V. Trishina, From the set of recurrent equations to the set of systolic wavefront algorithms, TR, Computing Center, Academy of Sciences, Novosibirsk, 1989.

[84] T. Toffoli, Cellular automata as an alternative to (rather than an approximation of) differential equations in modelling physics, in: *Cellular Automata, Proc. Interdisciplinary Workshop*, Los Alamos (North-Holland, Amsterdam, 1983) 117-127.

[85] P. J. Varman and I. V. Ramakrishnan, A fault-tolerant VLSI matrix multiplier, Dept. of Computer Science, TR-85-29, SUNY at Stony Brook, 1985.

[86] R. Vollmar, Some remarks on pipelined processing by cellular automata, *Comput. Artificial Intelligence* **6** (1987) 263-278.

[87] R. Vollmar, Basic research for cellular processing, in: *Proc. PARCELLA* (Akademie-Verlag, Berlin, 1988) 205-222.

[88] U. Weisser and A. Davis, A wavefront notation tool for VLSI array design, in: *VLSI Systems and Computations* (Computer Science Press, 1981) 226-234.

[89] W. P. Weijland, A systolic algorithm for matrix-vector multiplication, in: *Proc. Computing Science in The Netherlands* (1987) 143-160.

[90] S. Wolfram, Statistical mechanics of cellular automata, *Rev. Modern Phys.* **55** (1983) 601-644.

[91] S. Wolfram, Computation theory of cellular automata, *Comm. Math. Phys.* **96** (1984) 15-57.

[92] S. Wolfram, Universality, and complexity in cellular automata, *Physica* **10D** (1984) 1-35.

[93] E. Fachini, J. Gruska, A. Maggiolo Schettini and D. Sangiorgi, Simulation of systolic tree automata on trellis automata, TR, Dipartimento di Informatica, Universita di Pisa.

[94] K. Culik II, A. Salomaa and D. Wood, Systolic tree acceptors, *RAIRO Inform. Théor.* **18** (1984) 53-69.