11

Systematic Design Approaches for Algorithmically Specified Systolic Arrays

José A. B. Fortes
*King-Sun Fu**
Benjamin W. Wah

11.1. INTRODUCTION

The evolution in Very-Large-Scale-Integration (VLSI) technology has had a great impact on computer architecture (114). Many existing algorithms in pattern recognition and image and signal processing can be implemented on a VLSI chip using multiple, regularly connected processing elements (PEs) to exploit the great potential of pipelining and multiprocessing in applications in command, control, and communications systems (4). This type of array processor has been referred to as a *systolic array*, and the concept was introduced in the pioneering paper of H. T. Kung and C. E. Leiserson (67, 69). A good survey on the state of the art is provided by Ullman (116). A list of the sample problems for which systolic solutions exist is shown in Table 11.1 (46).

* Deceased.

Research supported by National Science Foundation Grants ECS 80-16580, DMC 85-19649, and DMC 84-19745, and was also supported by the Innovative Science and Technology Office of the Strategic Defense Initiative Organization and was administered through the Office of Naval Research under contract no. 00014-85-k-0588.

Table 11.1 A Sample of the Applications for Which Systolic Arrays Are Available

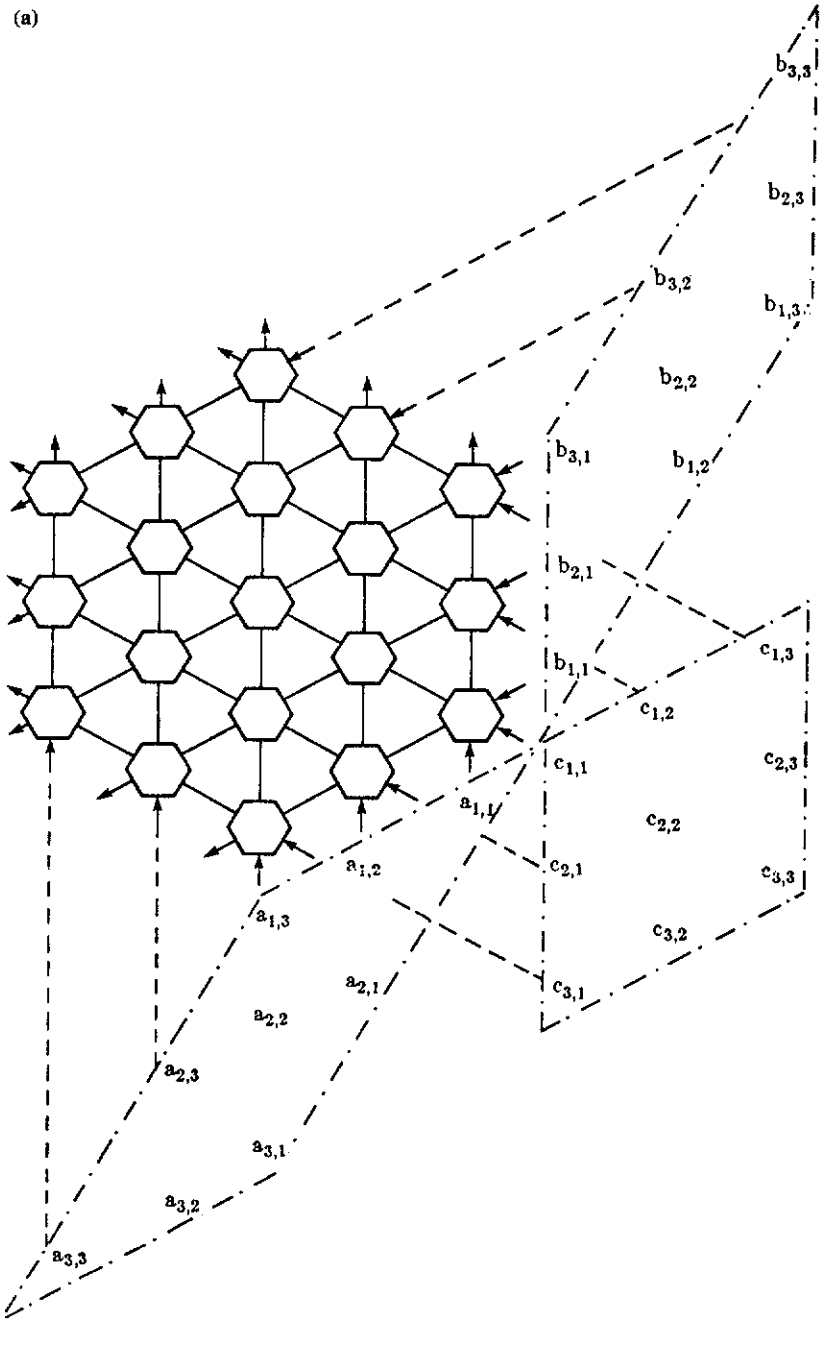
Signal and image processing and pattern recognition
- FIR, IIR filtering and 1-D convolution[21–24,33,57,69,82]
- 2-D convolution and correlation[18,64,65,73,77,92]
- discrete Fourier transform[9,21,22,57,58,69,71,123]
- interpolation[18]
- 1-D and 2-D median filtering[72]
- geometric warping[6,18,124]
- feature extraction[11]
- order statistics[72,82]
- counters[82]
- minimum distance classifier[12]
- covariance matrix computation[36]
- template and pattern matching[22,36,82]
- seismic signal classification
- cluster analysis[120]
- syntactic pattern recognition[54,89]
- radar signal processing[30,90,91]
- curve detection[103]
- dynamic scene analysis[31]
- image resampling[44]
- scene matching[30]

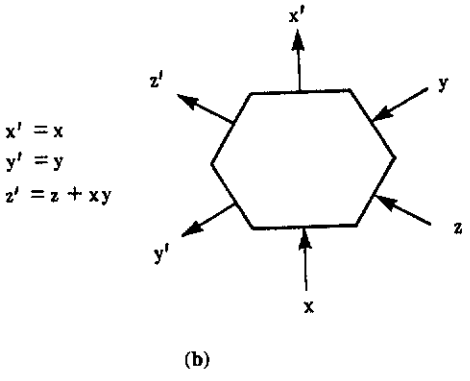
Matrix arithmetic
- Toeplitz matrix–vector multiplication[69]
- matrix–matrix multiplication[3,34,69,82,119]
- matrix triangularization[35,52,69]
- QR and LU decompositions[1,35,49,52,82,112,115]
- sparse-matrix operations[53]
- solution of triangular linear systems[1,49,82]
- polynomial multiplication/division[33]

Nonnumeric applications
- data structures—stacks and queues,[14] searching,[10,76,92,94] sorting and priority queues
- graph algorithms—transitive closure,[51] minimum spanning trees,[8] connected components[105]
- language recognition[7,50,82,107,109,111]
- dynamic programming[42,43,51]
- arithmetic arrays[2,3,45]
- relational database operations[10,32,51,66,113]
- algebra[51,93]

An example of a systolic array for multiplying matrices A and B to form C is shown in Figure 11.1. The dataflows of the three rhomboidal data blocks are in three directions: A moves toward the north, B moves toward $-120°$ north, and C moves toward $-60°$ north. During a clock cycle, each PE receives three data items from three different pipes and executes a multiply-add operation. These data items advance into neighboring PEs along their own pipes synchronously in the next clock cycle.

(a)


b3,3
b2,3
b3,2
b1,3
b2,2
b3,1
b1,2
b2,1
b1,1
c1,3
c1,2
c1,1
c2,3
a1,1
c2,2
a1,2
c2,1
c3,3
a1,3
c3,2
a2,1
c3,1
a2,2
a2,3
a3,1
a3,2
a3,3



(b)

Figure 11.1 The systolic processor for two-dimensional matrix multiplication. (a) Systolic processor. (b) Structure of PE.

One of the many advantages of the systolic approach is that each input-data item can be used a number of times once it is accessed, and thus a high computational throughput can be achieved with only a modest bandwidth. Other advantages include modular expandability, simple and regular data and control flows, and simplicity and uniformity of PEs.

Systolic arrays have been classified into semisystolic arrays with global data communications and pure systolic arrays without global data communications (67). In a *semisystolic array,* a data item accessed from memory is broadcast to and used by a number of possibly nonidentical PEs concurrently. Although this approach is potentially faster than systolic arrays without data broadcast, providing (or collecting) a data item to (or from) all the PEs in each cycle requires the use of a global bus that may eventually slow down the processing speed as the number of PEs increases. On the other hand, a *pure systolic array* eliminates the use of broadcast buses and implements the algorithm in pipelines extending in different directions. Several data items flowing along different pipes with the same or different rates may meet and interact. The PEs operate synchronously with one or more clocks, and all the necessary operands to be processed by a PE in each computational step must arrive at this PE simultaneously. This mode of pipelining is referred to as *systolic processing*.

One of the important design problems in systolic processing is the development of a systematic methodology for transforming an algorithm represented in some high-level constructs into a systolic architecture specified by the timing of data movements and the interconnection of processing elements such that the design requirements are satisfied. In this chapter, we survey 19 methodologies proposed in the literature. The applicability, capabilities, and results derived for each methodology are identified.

11.2. Systematic Methodologies for Designing Systolic Arrays

The common characteristic of most previously proposed methodologies is the use of a transformational approach—i.e., systolic architectures are derived by transforming the original algorithm descriptions that are unsuitable for direct VLSI implementation. Distinct transformational systems for systolic-architecture design (hereafter referred to as transformational systems) can be characterized by how algorithms are described, what formal models are used, how systolic architectures are specified, and what types of transformations are used on and between these respresentations. In other words, we can visualize each transformational system as a three-dimensional space, where dimensions (or axes) are associated with the algorithm representation, algorithm model, and architecture specification. To the axis of algorithm representation, we associate different forms or levels to present an algorithm to the transformational system. The axis of algorithm model shows different levels of abstraction used to represent relevant features of the algorithm. The axis of architecture specification is associated with the hardware model or level of design in which the systolic array is described.

This three-dimensional space can be graphically depicted as a Y chart (Figure 11.2), where directed arcs can be drawn to illustrate transformations that map a given representation into another representation in the same axis and level (a self loop), in the same axis and different level, or between distinct dimensions.[1] Arcs drawn in full lines represent systematic transformations, whereas those drawn in broken lines represent ad hoc transformations. The Y charts allow us to classify and describe the large number of approaches taken to design systolic arrays. Before we do this, we will use the Y chart in Figure 11.3 to explain Kuhn's approach (63).

Kuhn's methodology starts with a naive high-level language cyclic-loop program—i.e., an algorithm written without regard to how it is implemented in VLSI. In an ad hoc manner, additional subscripts for variable referencing are introduced such that the possibility of broadcasts of variables does not exist. The algorithm model assumed in Kuhn's method is a set of computation nodes (which correspond to the loop-body assignment statements) indexed by the vector value of the indices of the iteration when they are computed (Figure 11.3). The structural information is modeled by the dimensionality of the iteration space and the dependency vectors (which are the vector difference of the indices of dependent computation nodes). The geometry of the algorithm is represented by the iteration space and how different variables are associated to points in that space. This model is derived from the program in a systematic manner by using analysis techniques

[1] We borrowed from Gajski's paper (47) the idea of using Y charts to improve the clarity of our presentation; however, our Y charts are minimally related to those used in that reference.

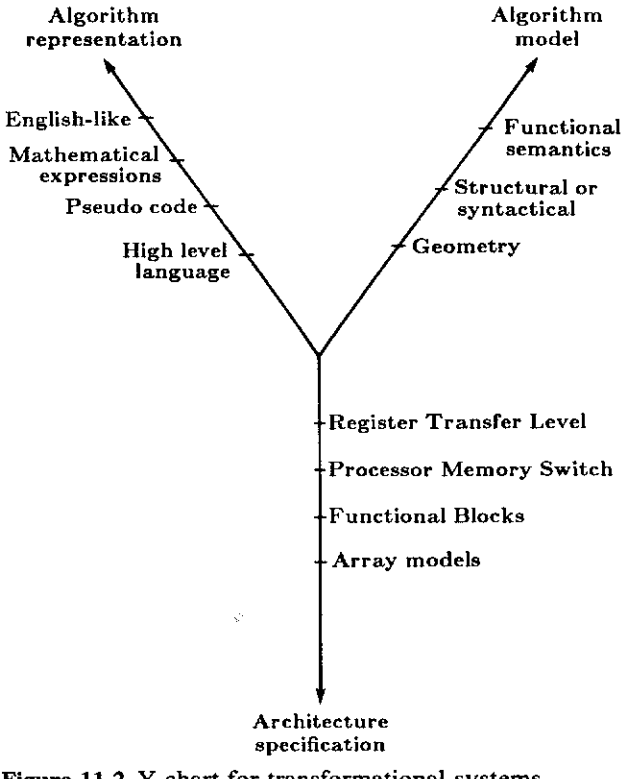


Figure 11.2 Y-chart for transformational systems.

similar to those used in optimizing compilers. A re-indexing transformation is then sought in an ad hoc fashion until a favorable set of dependencies is obtained. Once this transformation is known, one can systematically generate not only the new dependency vectors but also the range of the new indices of the loops and the subscript functions used to reference variables. By projecting the new iteration set into all but one of its dimensions, and by identifying the iterations in which input variables are used, the size, dimension, and input/output ports of the architecture can be systematically generated.

Each point in the projected space corresponds to a PE in the array whose function is totally described by the statements in the loop body. The interconnections and the direction, speed, and timing of data movements are systematically derived from the new set of dependencies that resulted from applying the re-indexing transformation. This completes our example of the use of Y-charts to explain a methodology. We defer the analysis of the capabilities of Kuhn's method until after we introduce a classification of the different approaches in terms of the Y-charts.

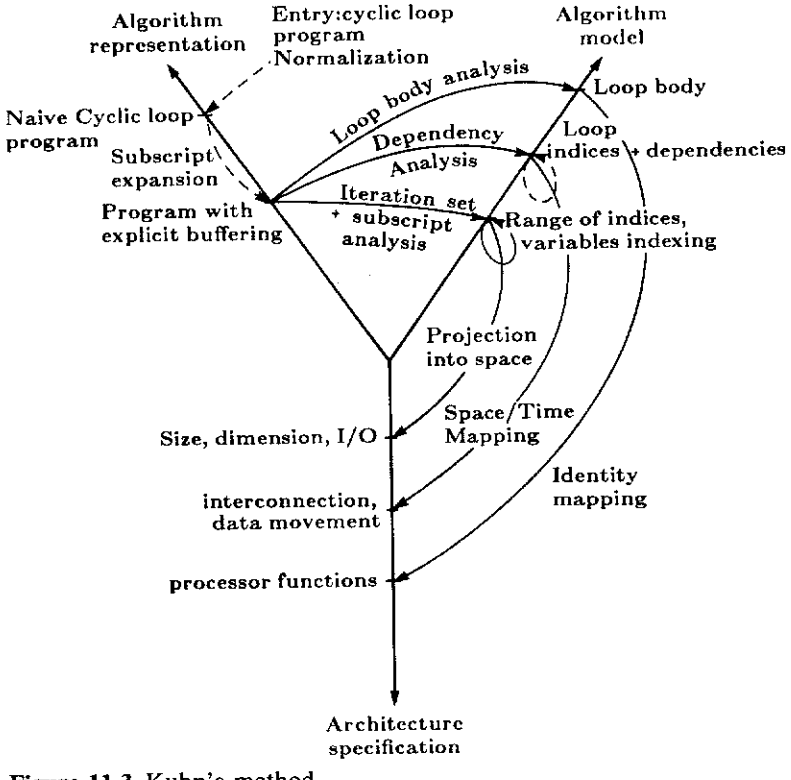


Figure 11.3 Kuhn's method.

The various methodologies can be grouped into the following classes:[2]

1. Those that allow transformations to be performed at the algorithm-representation level and that advocate a direct mapping from this level to the architecture specification. These include:
   Cohen, Johnsson, Weiser, and Davis' method
   Lam and Mostow's method
2. Methods that prescribe transformations at the algorithm-model level and that require procedures for deriving the model from the algorithm representation (analysis) and for mapping the model into hardware (synthesis). These include:
   Gannon's method
   H. T. Kung and Lin's method
   Kuhn's method
   Moldovan and Fortes' method

[2] The order in which the methodologies are described is chosen at random.

Miranker and Winkler's method
Cappello and Steiglitz's method
S. Y. Kung's method
Quinton's method
Ramakrishnan, Fussell, and Sillberschatz's method
Li and Wah's method
Cheng and Fu's method
Jover and Kailath's method
Schwartz and Barnwell's method
Ibarra, Kim, and Palis' method

3. Methods that transform a previously designed architecture into a new architecture. We will consider only one work in this class:
   Leiserson, Rose, and Saxe's method
4. Methods that abstract the function implemented by a given systolic architecture and that use symbolic manipulations and transformations to prove the correctness of the design. Two studies in this class are considered:
   Chen and Mead's method
   Kuo, Levy, and Musicus' method

In the following sections, we will describe these methods in an arbitrary order, show their applicability, discuss their capabilities, and summarize the major results. The discussion in some of these studies may be vague, and we have tried to infer their results from our understanding of the published work.

11.2.1. Cohen, Johnsson, Weiser and Davis' Method (33, 57–59, 119)

Description

Starting from a mathematical expression involving subscripted variables, which conceptually represent data sequenced in time or space, this method begins by deriving a new expression where a well-defined operator Z is used to model displacements in time (e.g., the storage of data) or shifts in space (e.g., the allocation of a data stream to PEs). Symbolic manipulation is used to transform the derived mathematical expression into equivalent ones by using the properties of the Z operator and the functional operators in the expression. From a particular expression, the execution order of the operations can be derived from known precedence rules. The number, placement, and interconnection of operator PEs can also be derived. Timing and storage requirements are inferred from the placement of delay PEs (which correspond to the Z operators) (Figure 11.4).

Applicability

This method seems to be best applicable to algorithms that can be described by relatively simple and concise mathematical expressions.

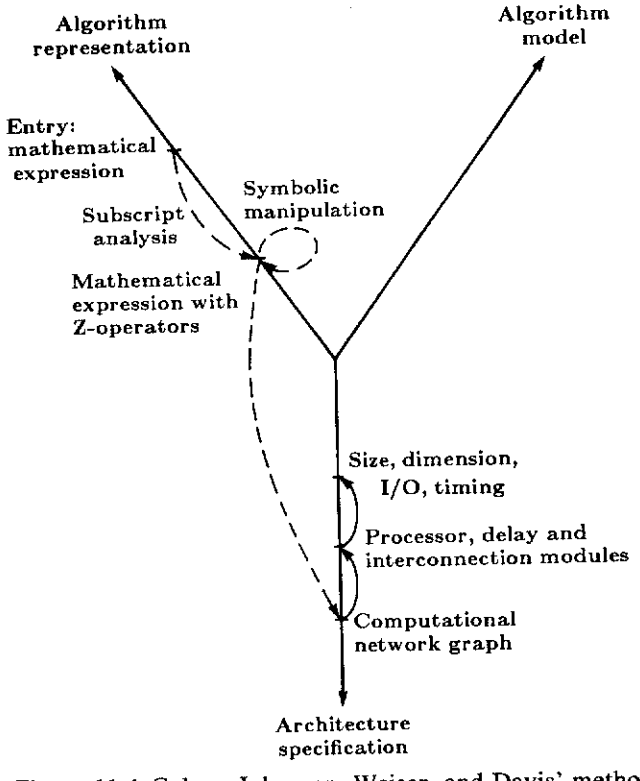


Figure 11.4 Cohen, Johnsson, Weiser, and Davis' method.

Capabilities

Computational rate, performance, delay, modularity, and size can be easily derived from the equations; interconnection and communication characteristics can also be derived when the architecture is regular. This method may yield implementations with both parallel and sequential features, requiring hardware of a size smaller than the problem size. The method treats control signals in the same way as data signals. The optimal design is searched in an ad hoc manner.

Results

Formal derivations have been reported for architectures intended for the following problems: finite-impulse-response (FIR) filters, discrete Fourier transform (DFT), matrix-vector product, string matching, solution of triangular linear equations, product of band matrices, synthetic aperture radar (SAR), and multiplication and division of polynomials. A set of data-set operators defined in terms of the Z operator was also proposed for treating

sets of data as wavefront entities in expressions and their graphical representation.

11.2.2. Lam and Mostow's Method (SYS) (78, 101)

Description
SYS accepts as input an algorithm suitable for systolic implementation—i.e., an algorithm obtained by software transformations from a high-level specification that results in segments of code executed repeatedly with a regular pattern of data accesses. The algorithm is mapped into a systolic design described by a structure and a driver. The structure describes the hardware PEs (which are functionally equivalent to code segments), interconnections, and input–output ports. The driver defines data streams in terms of the original variables in the algorithm. The mapping of iterative algorithms uses three basic allocation schemes named sequential, parallel, and compositional. SYS has a special language for representing a given design. Initially, SYS generates a simple-minded implementation of the given algorithm. Systematic and user determined transformations are then used to optimize and to obtain new designs (Figure 11.5).

Applicability
SYS can process algorithms with simple FOR loops and BEGIN-END blocks, simple unnested function calls, and scalar and array variables. As reported in the references, SYS cannot deal with conditional execution, computed iteration bounds and array indices, and other high-level software constructs.

Capabilities
SYS can derive the structure and driver of a systolic design. Specification of the structure includes the number and dimensionality of ports of PEs, hierarchical definition of PEs, arrays and compounds of PEs, and interconnections among them, including broadcasts and directional links. The driver describes data streams and timing schemes that include delay, skew of streams, and ready time (time allowed between two consecutive inputs to the structure).

Results
Reported designs obtained by SYS include two systolic arrays for polynomial evaluation and a circuit for computing the greatest common divisor of two polynomials. Other nonsystolic designs using a transformational system related to SYS include a chip for color shading and hidden-surface elimination and a multichip switching network for marker passing semantic networks. All designs were previously known.

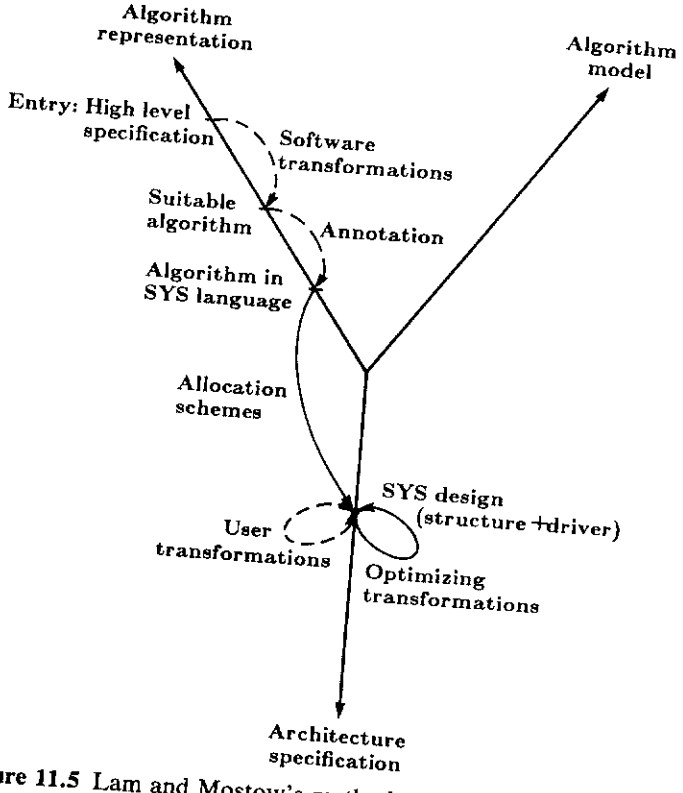


Figure 11.5 Lam and Mostow's method.

11.2.3. Gannon's Method (48)

Description

From a given algorithm, a functional specification is derived by using vector operators that explicitly represent parallelism. These vector operators are defined in terms of basic functions that correspond to small units of sequential computation and that map directly into the functional specification of the PEs of the systolic architecture. The vector operators include a product operator, which represents the concurrent operation of basic functions, permutation and data-movement operators; a chain operator, which represents the iterative composition of basic functions; and the systolic-iteration operator, which describes basic functions that are "reused." The global functional specification of the algorithm is viewed as a dataflow graph which, depending on the properties of the functions and operators used, can be mapped into a systolic architecture. Different architectures result from expressing the same algorithm with different operators (Figure 11.6).

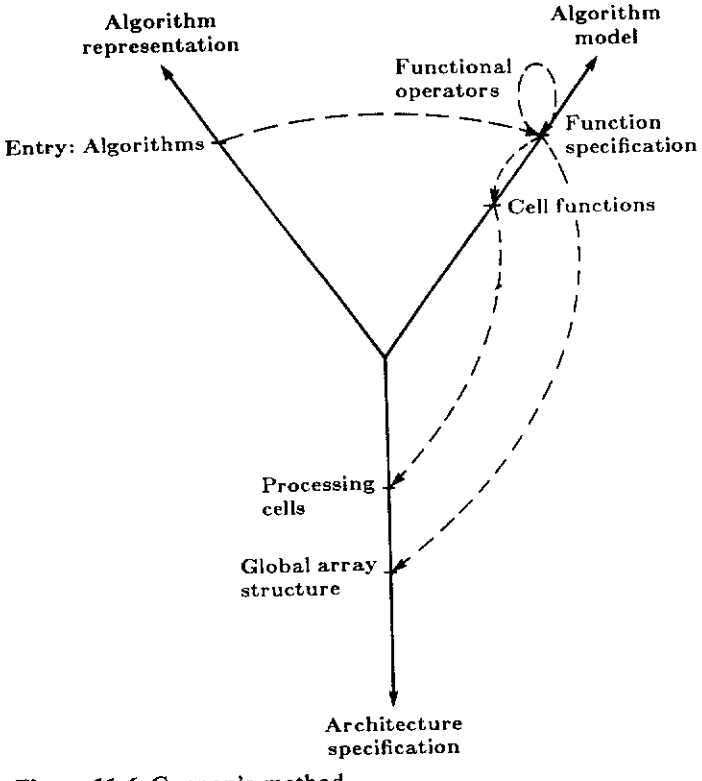


Figure 11.6 Gannon's method.

Applicability
This method seems to be suitable to those algorithms that can be reexpressed by vector operators. For these algorithms, the methodology seems hard to apply without human assistance.

Capabilities
The functional description of PEs and the interconnection topology can be easily derived. Additional information such as data movement and timing is present in implicit form.

Results
A previously known design of a recurrence solver was rederived. The formalism used proved the theoretical result that systolic versions of computation graphs perform asymptotically as fast as fully concurrent execution of the original dataflow graph.

11.2.4. H. T. Kung and Lin's Method (70)

Description

This method starts by deriving a straightforward and obviously correct algebraic representation from the mathematical representation of the algorithm. The canonical algebraic representation consists of two matrical expressions of the form (a) $v \leftarrow Av + bx$, and (b) $y = c^T v$, where x represents the input, y represents the output, and v represents variables generated by implicit functions. The $(n \times n)$ matrix A and the column vectors b and c represent the delay cycles between the availability and the use of variables, and each entry is either 0 or Z^{-k}, where k corresponds to the number of delays. For example, the i-th component of v in Expression (a) is

$$v_i \leftarrow Z^{-a_{i,1}} v_1 + \cdots + Z^{-a_{i,n}} v_n + Z^{-b_i} x,$$

which means that

$$v_i(t) = f_i[v_1(t - a_{i,1}), \ldots, v_n(t - a_{i,n}), x(t - b_i)]$$

for some implicit function f_i associated with node v_i. To this canonical representation, algebraic transformations are then applied. There are two major types of transformations, retiming and "k-slowing," which can also be described algebraically. These transformations determine the distribution of delays and the input/output periods of the systolic architecture. There exists a direct correspondence between the algebraic representations and a hardware-related representation denoted as the Z-graph. The Z-graph has an edge for each variable and a node for each computation. Each edge is labeled by Z^{-k} if k delay cycles (i.e., registers) exist between the availability of the variable and its use as an operand or output (Figure 11.7).

Applicability

The method is suitable for algorithms for which a canonical algebraic representation can be found.

Capabilities

Functional description of PEs, interconnections, and timing for input/output and data communication can be derived systematically; designs and transformations can be expressed algebraically; theoretical results on retiming and k-slowed designs can be proved easily by algebraic manipulation.

Results

Designs for FIR and IIR filters and matrix–matrix multiplication were derived. New results include the derivation of two-level pipelined systolic arrays and systolic architectures for LU decomposition.

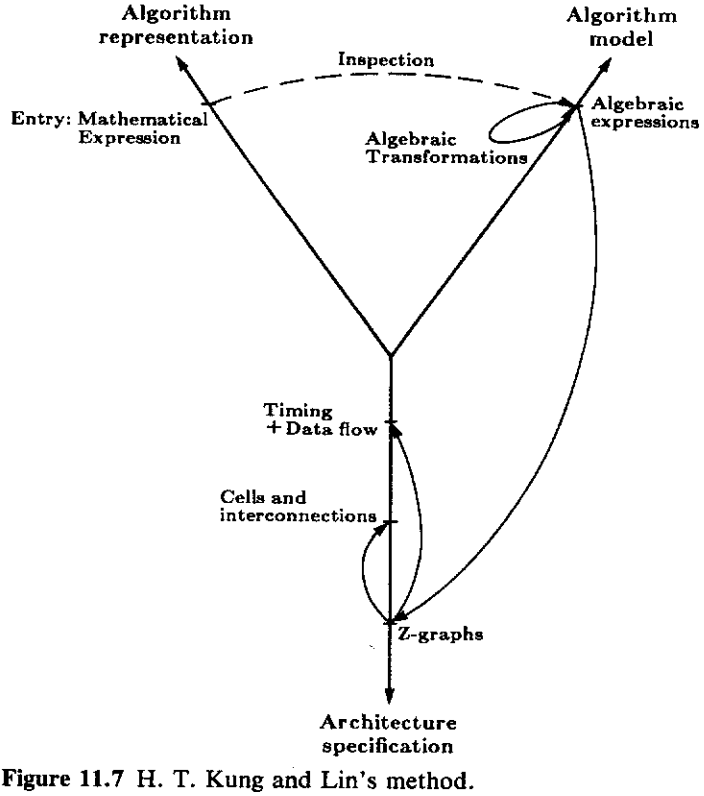


Figure 11.7 H. T. Kung and Lin's method.

11.2.5. Kuhn's Method (62, 63)

Description
As described early in this chapter.

Applicability
This method is best suited for algorithms described as cyclic-loop programs with constant execution time and dependencies in loop bodies.

Capabilities
Size, dimension, topology, input–output ports, execution time, data movement, timing, and functional descriptions of PEs of an architecture can be systematically derived. However, nothing can be said about the optimality of the design, and the choice of transformations is done in an ad hoc manner.

Results
Designs for the following problems were derived using this method: matrix–matrix multiplication, matrix–vector multiplication, recurrence evaluation, solution of triangular linear systems, constant-time priority queue, on-line sort, transitive closure, and LU decomposition.

11.2.6. Moldovan and Fortes' Method (37–41, 96–99, 100, 104)

Description
From a program or a set of recurrence equations, an algebraic model of the algorithm is derived by using systematic techniques similar to those used in software compilers. This model consists of a structured set of indexed computations that operate on a set of inputs to obtain a set of outputs. Typically, programs include loops, and indexing of computations is related to the loop indices. However, unlike Kuhn's approach, which associates the body of loops with the corresponding loop indices, each computation has an index. The algebraic representation of the algorithm is then transformed by local and global transformations. Local transformations are used to rewrite computations that are mapped into the functional and structural specifications of the PEs of the systolic architecture. Global transformations composed of time and space transformations are used to restructure the algorithm. They are chosen in such a way that the new algorithm has a set of dependencies that favors VLSI implementation. Time transformations determine the execution time of the algorithm and the timing for data communication. Space transformations determine the interconnections and the direction of data movement. The projection of the index set of the algorithm into space determines the size, dimension, I/O ports, and geometry of the architecture (Figure 11.8).

Applicability
This method is best suited to algorithms described by either programs with loops or recurrence equations.

Capabilities
Because this method is an extension of Kuhn's approach, it has the same capabilities as that method. Additionally, it allows or eliminates broadcasting, designs fixed-size architectures for arbitrarily large algorithms, implements fault-tolerance schemes, and optimizes execution time.

Results
Systematically obtained designs have been reported for matrix–matrix multiplication, LU Gaussian elimination, dynamic programming, partitioned matrix–vector multiplication, convolution, partitioned QR-eigenvalue de-

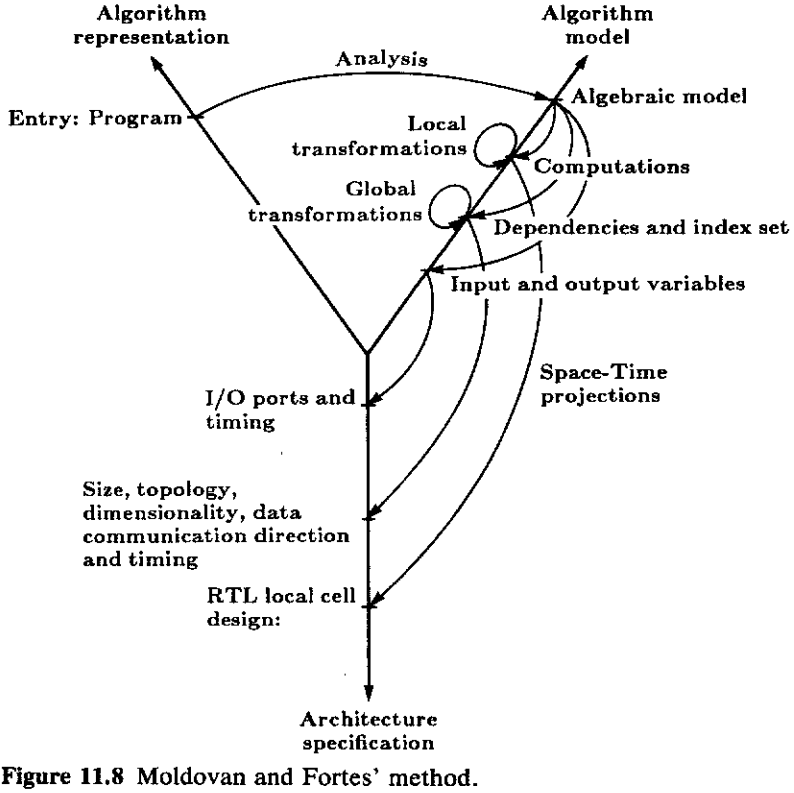


Figure 11.8 Moldovan and Fortes' method.

composition, and partial differential equations. Theoretical results include the necessary and sufficient conditions for the existence of global transformations and broadcasts, sufficient conditions for the partitionability of algorithms, and a method for finding optimal linear schedules for systolic algorithms.

11.2.7. Miranker and Winkler's Method (95)

Description

This method is an extension of Kuhn's method and is similar to Moldovan's method. An algorithm is represented as either a mathematical expression or a cyclic-loop program. One extension is to allow the rewriting of mathematical expressions by using the properties of the operators in an ad hoc fashion. The other extension is to use the graph embeddings based on the knowledge of the longest path of the computation graph when this graph is too irregular and simple matrix transformations are not useful (Figure 11.9).

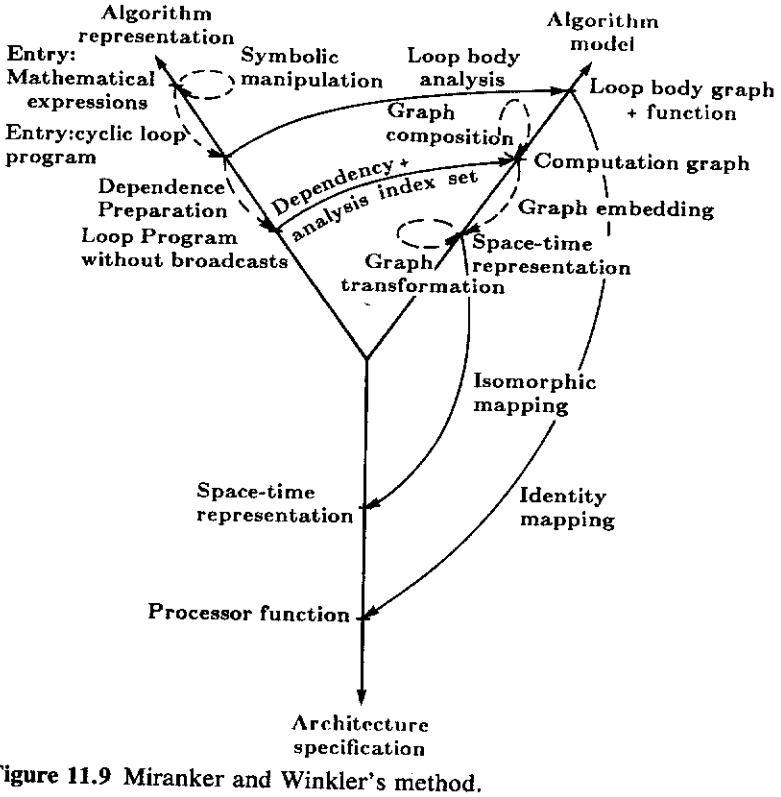


Figure 11.9 Miranker and Winkler's method.

Applicability
Theoretically, it can be applied to any algorithm, although systematic design seems possible only for those algorithms described by programs with loops.

Capabilities
Size, dimension, topology, data movement, timing, functional description of PEs, and execution time can be systematically derived.

Results
Designs of architectures for the computation of discrete Fourier transform and the solution of a triangular linear system of equations were systematically derived.

11.2.8. Cappello and Steiglitz's Method (15–17)

Description
Starting from a set of recurrence equations describing the algorithm, a canonical representation is obtained by adjoining an index representing time

to the definition of the recurrence. Each index is associated with a dimension of a geometric space, where each point corresponds to a tuple of indices on which the recurrence is defined. To each such point, a primitive computation is associated, and its implementation is left unspecified. Primitive computations are mapped directly into functional specifications of PEs in the systolic architecture. From the geometric representation and an ordering rule, the topology and size of the architecture and the timing and direction of dataflows are derived systematically. By selecting different geometric transformations, distinct geometric representations and their corresponding architectures can be derived (Figure 11.10).

Applicability
This method is best suited to algorithms described by recurrence equations.

Capabilities
Geometric representations help the designer's understanding of a systolic architecture, and geometric transformations are easily and succinctly rep-

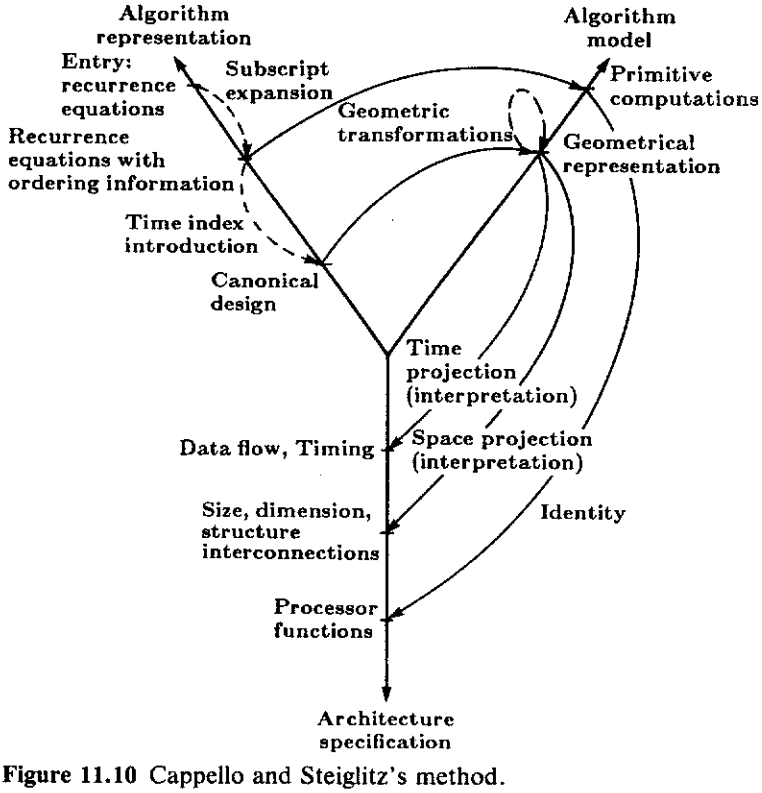


Figure 11.10 Cappello and Steiglitz's method.

resented as matrix transformations. The presence of broadcasts, data pipelining, topology, area, and timing of the architecture are easily perceived from the geometric representation.

Results
Designs of architectures for matrix–vector multiplication, convolution, matrix–matrix product, and matrix transposition were formally related and rederived. For some, it was shown that they are asymptomatically optimal, and for others, alternative designs were provided.

11.2.9. S.Y. Kung's Method (74, 75)

Description
Given a Signal Flow Graph (SFG) representing an algorithm, this method starts by choosing basic operational modules that correspond to the functional description of PEs of the architecture. Localization rules are then applied to derive a regular and temporally localized SFG. The localization procedure consists of selecting cut-sets of the SFG and reallocating scaled delays to edges "leaving" and "entering" the cut-set in such a way that at least one unit of time is allowed for communicating a signal between two nodes. Delays are combined with operational modules to obtain a full description of the operation of a basic systolic module. The resulting SFG maps straightforwardly into the systolic array by mapping basic modules into PEs and edges into interconnections. Timing and data movements can be derived from the basic modules due to the localized spatial and temporal characteristics of the SFG (Figure 11.11).

Applicability
This method is applicable to all algorithms described by computable SFGs with some regularity.

Capabilities
Size, dimension, functional, and structural description of PEs, timing, direction of dataflow, and interconnections of the architecture can be derived from the SFG of the algorithm. Design verification can be done by applying Z-transform techniques to the SFG.

Results
Systolic arrays have been derived from SFGs for autoregressive filter, matrix multiplication, banded-matrix–full-matrix product, banded-matrix multiplication, and LU-matrix decomposition. Theoretical results include the proof that all computable SFGs are temporally localizable and the equivalence between SFGs and dataflow graphs.

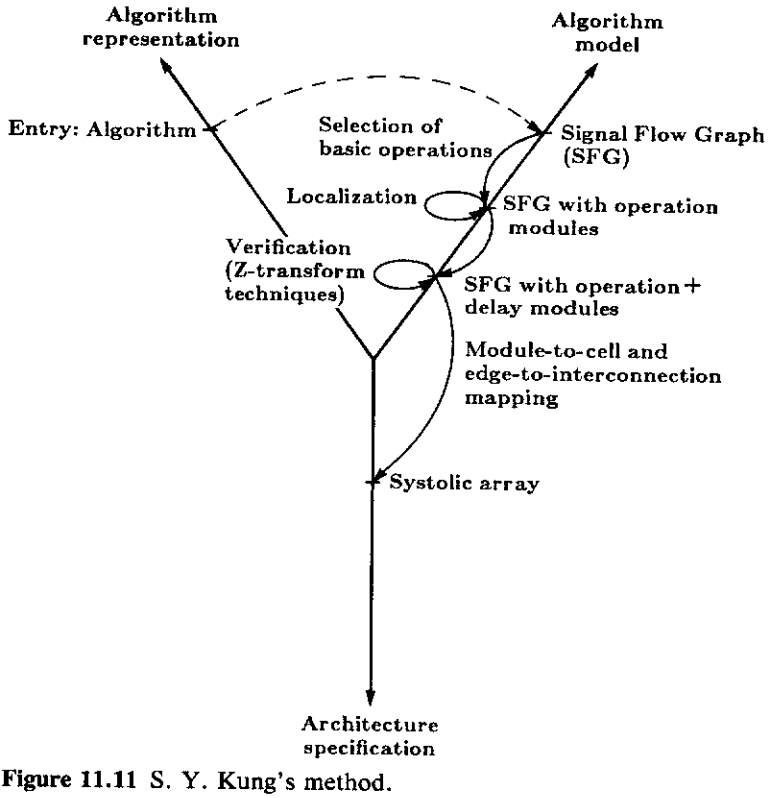


Figure 11.11 S. Y. Kung's method.

11.2.10. Quinton's Method (106, 107)

Description

Given a system of n uniform recurrence equations defined over some convex subset D in Z^M and with some characteristic dependency vectors, (which, together define a dependency graph), this method starts by finding a timing function that maps points of D into time. This requires the identification of a convex space of feasible timing functions from which one can be chosen heuristically. Such space can be found systematically from the knowledge of the dependency vectors and D(D can be thought of as the index set of the recurrence). Next, an allocation function is chosen, which projects D into space along a preselected direction such that two points in D with the same image under the timing function do not map into the same point in space. Once the timing and allocation functions (which are quasi-affine functions) are known, the systolic array can be systematically generated from D. Each point of D is mapped into a PE that computes the recurrence function, and receives and sends data from and to PEs that are the image of

points dependent and depending on the point under consideration, with delays given by the timing function (Figure 11.12).

Applicability
The method is specifically intended for algorithms described by uniform recurrence equations.

Capabilities
The functional description of PEs, the size, dimension, and topology of the array, and the execution time, direction, and timing of data communication can all be derived systematically.

Results
Derived architectures include arrays for convolution (including a block convolver and a ring convolver) and matrix product. Extensions of the method allow the derivation of arrays for LU decomposition and dynamic programming.

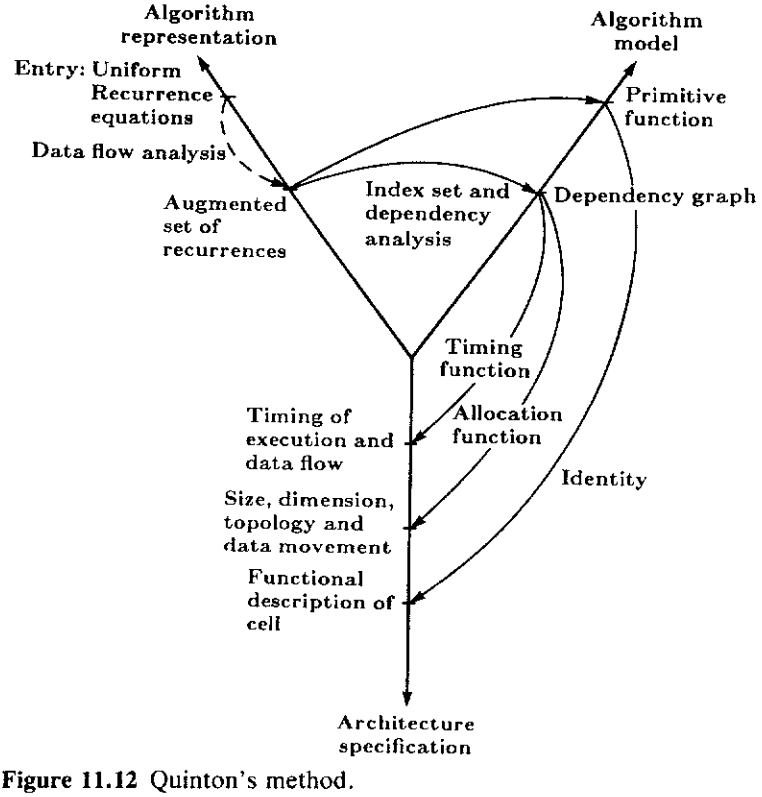


Figure 11.12 Quinton's method.

11.2.11. Ramakrishnan, Fussell, and Sillberschatz's Method (108, 118)

Description

This method starts with a dataflow description of the algorithm (an acyclic program graph), which is partitioned into sets of vertices that are mapped into the same PE (this partitioning is called diagonalization). A syntactically correct mapping is then used to map computation vertices onto PEs and time steps, and the labels and edges to map communication delays and interconnections (Figure 11.13).

Applicability

The method applies only to homogeneous graphs with connectged subgraphs that satisfy certain properties. Moreover, the method can only be used to generate linear arrays.

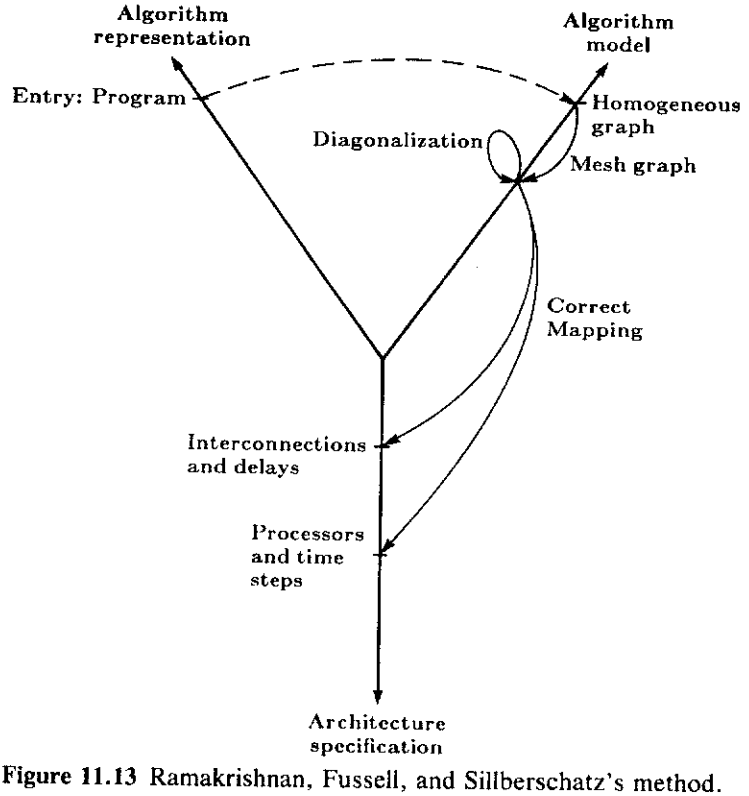


Figure 11.13 Ramakrishnan, Fussell, and Sillberschatz's method.

Capabilities
The method yields the number of PEs, their functional description, and the number of I/O ports. Additionally, it gives the direction of data communication, time used, and timing.

Results
Linear-array designs were synthesized for band matrix–vector multiplication, convolution, dynamic programming, and transitive closure.

11.2.12. Li and Wah's Method (85–88)

Description
Starting from an algorithm described as a set of linear recurrence equations, this method derives three classes of parameters: velocities of dataflows, spatial distributions of data, and periods of computations. The relationships among these parameters are represented as constraint vector equations that must be satisfied in a correct design. The performance of a design can also be expressed in terms of the defined parameters. Performance can be defined as execution time or the product of the square of execution time and the number of PEs. Optimal designs are then searched in the space of solutions that satisfy the constraint equations. This search is done by ordered enumeration over a limited search space in time polynomial to the problem size. The functional description of the PEs is derived from the definition of the recurrence equations. The interconnections among PEs are found from the defined parameters (Figure 11.14).

Applicability
The method is best suited to algorithms that can be described by sets of linear recurrences.

Capabilities
Functional description of PEs, timing and spatial distribution of dataflows, execution time, number of PEs, and interconnections can be systematically derived. Optimal designs can be found.

Results
Systematically derived architectures include systolic arrays for finite-impulse-response (FIR) filtering, matrix multiplication, discrete Fourier transform, polynomial multiplication, deconvolution, triangular matrix inversion, and tuple comparison.

11.2.13. Cheng and Fu's Method (26–30)

Description
Starting from a recursive formula with several indices or a program-loop with a simple expression, this method starts by designing the basic PE to

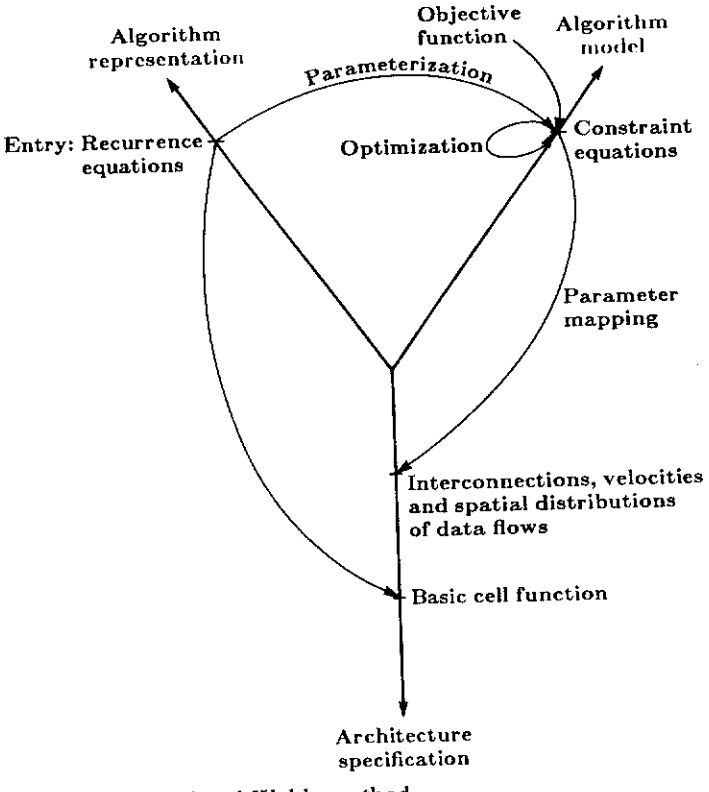


Figure 11.14 Li and Wah's method.

compute the simple expression. This basic PE is then expanded in time and space in such a way that indices of the loop (or recursion) become associated with time and space. This expansion is done according to rules that maintain the consistency of time and space. Time-space expansion can be applied in any degree varying from full-time expansion (i.e., purely sequential single-processor architecture) to full-space expansion (i.e., fully parallel single-time execution). Time and space expansions implicitly determine dataflow timing and direction as well as PE interconnections (Figure 11.15). The method can be implemented at the gate level, register level, processing-unit level, and system level. Because there is no restriction on the dimensionality of the processing array, the method can be applied to design high-dimensional VLSI architectures. A computational model and partition rules can be derived that partition any problem suitable for the method and implement the problem on a fixed-size VLSI architecture.

Applicability

The method is suitable for algorithms described by recurrence expressions or programs with loops.

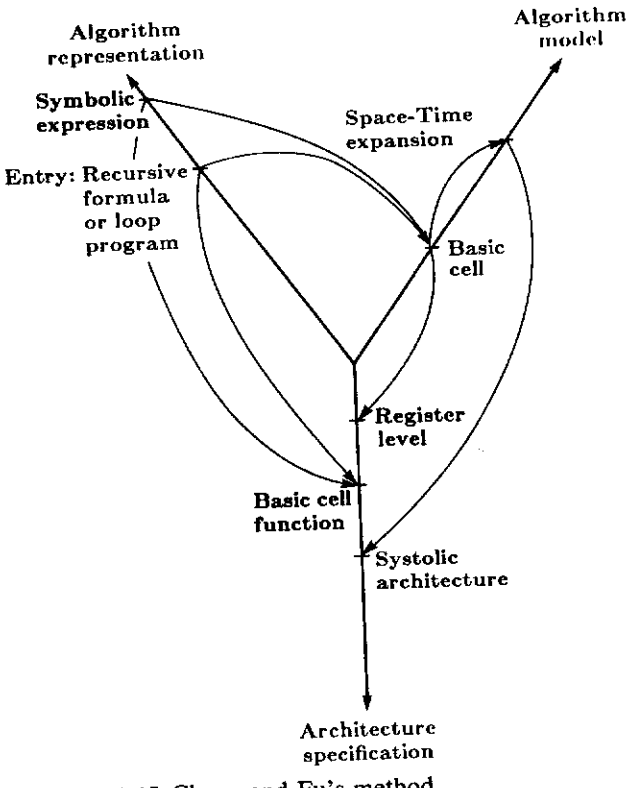


Figure 11.15 Cheng and Fu's method.

Capabilities
Functional description of PEs, timing and directions of dataflows, interconnections, execution time, and number of PEs can be systematically generated.

Results
This method has been applied to construct computational structures for computing vector inner product, matrix multiplication, convolution, comparison operations in relational databases, fast Fourier transform, hierarchical scene matching, transitive closure, string matching, pattern matching, recognition of hand-written signals, and recognition of context-free languages.

11.2.14. Jover and Kailath's Method (60)

Description
This method is based on the use of Lines Of Computation (LOCs) to determine whether a given topology is suitable for VLSI implementation. LOCs

are directional straight lines with several equally spaced nodes, and can be interpreted either as the history of how a given value was computed or as a stream of values in different stages of a computation. Because of the properties of LOCs, one can easily check if the LOCs chosen for a given algorithm define a systolic array or a systolic-type array (not necessarily planar and fully regular) (Figure 11.16).

Applicability
Suitable for algorithms from which one can easily identify LOCs.

Capabilities
The topology of the systolic array can be easily derived from LOCs. Additionally, throughput, efficiency, data interval, initial conditions, interval and external delays, and pipeline ability can also be found from LOCs and the knowledge of execution times of basic operations.

Results
Three designs for matrix multiplication were derived.

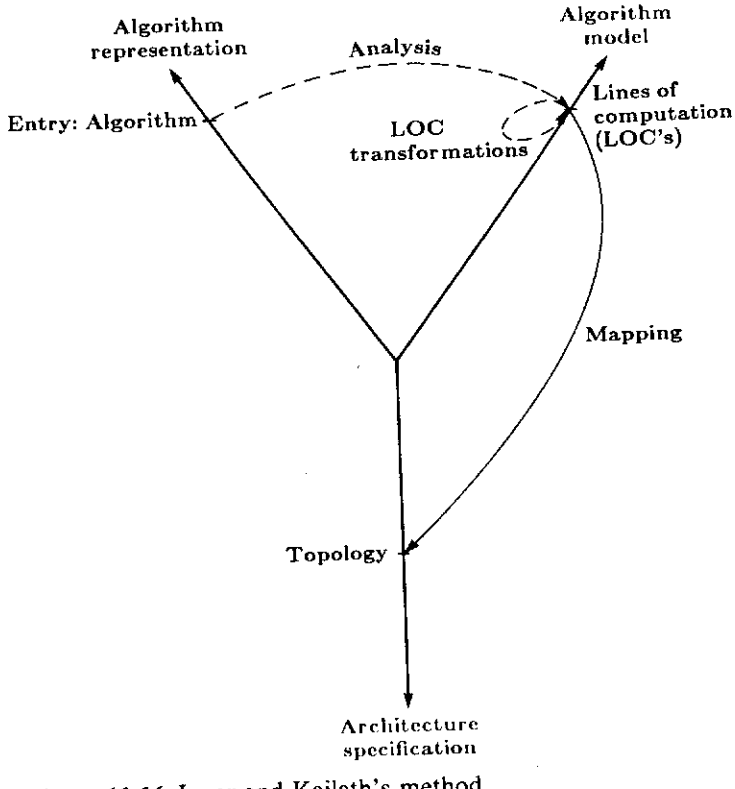


Figure 11.16 Jover and Kailath's method.

11.2.15. Schwartz and Barnwell's Method (5, 110)

Description

This method starts with an algorithm described as a fully specified flow graph—i.e., a directed graph in which nodes represent operations and edges represent signal paths. Node operations are fundamental operations performed by the PEs of the architecture. For a given flow graph, it is possible to derive a bound in the sampling period and a bound in the static-pipeline sampling period (i.e., the minimum sampling period achievable if the graph is implemented as a static pipeline). Different systolic solutions are generated by distributing delay nodes throughout the flow graph such that correctness is preserved and data transfers can be simultaneous. The transformations of flow graphs consist of data interleaving and the cut-set of delay transformation that are shown to preserve equivalence. The transformed flow graph is mapped into a systolic array by mapping nodes into PEs and delays and edges into interconnections (Figure 11.17).

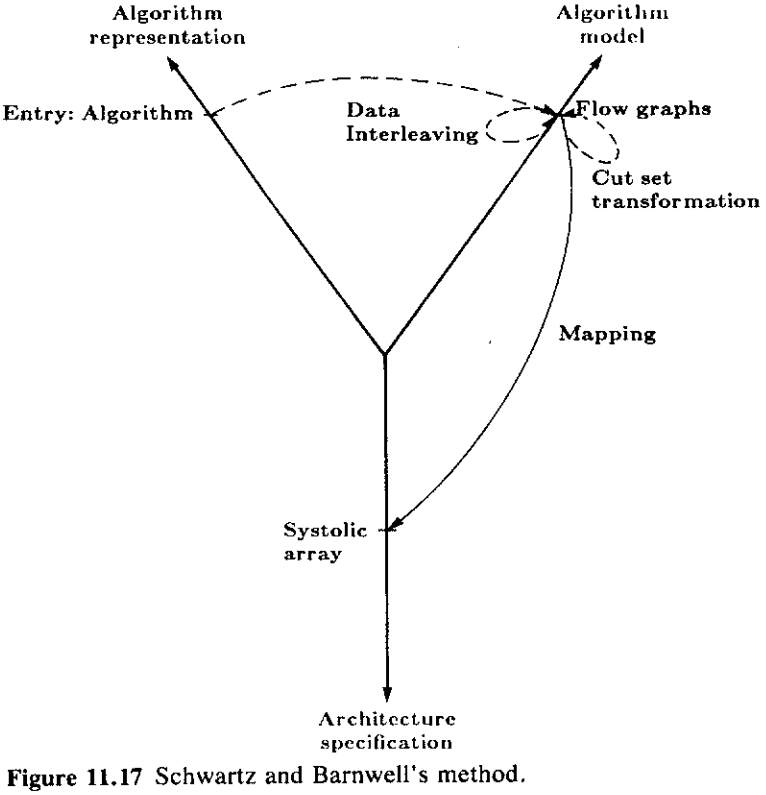


Figure 11.17 Schwartz and Barnwell's method.

Applicability
This method can be used with algorithms representable as shift-invariant flow graphs.

Capabilities
The following information about the systolic architecture can be systematically derived: number and functional description of PEs, interconnections, data movement, and time. The optimality of the resulting designs can be analyzed.

Results
Previously known and new architectures have been derived for FIR and IIR filters and two multiplier Markel–Gray lattice filters.

11.2.16. Ibarra, Palis, and Kim's Method (55, 56)

Description
Sequential-Machine (SM) models are used to simulate linear and orthogonal Systolic Arrays (SAs). Given an algorithm, this methodology starts by generating an SM characterization that consists of a serial program for a simple sequential machine with a single PE and an infinite array of registers (called the "worktape(s)"). SM characterizations that are obtained heuristically are easier to program and analyze than their SA counterparts. The conversions from SM to SA characterizations and vice versa are done systematically (Figure 11.18).

Applicability
This method seems to be best applicable to algorithms that can be easily programmed in an SM model. These are likely to be relatively simple and regular algorithms.

Capabilities
Because one starts with a given type of systolic array, certain features of the architecture (e.g., number, placement, direction of interconnections, and inputs) are known beforehand. In this sense, the methodology synthesizes or maps a given algorithm into a type of systolic array. From the SM program, the functional description of the PEs and the direction and timing of data movement can be derived automatically. The search and selection of the best type of systolic array and the best SM algorithm is done in an ad hoc manner.

Results
Systolic architectures and algorithms have been reported for priority queues, real-time bitwise multiplication, and language recognition. For linear systolic

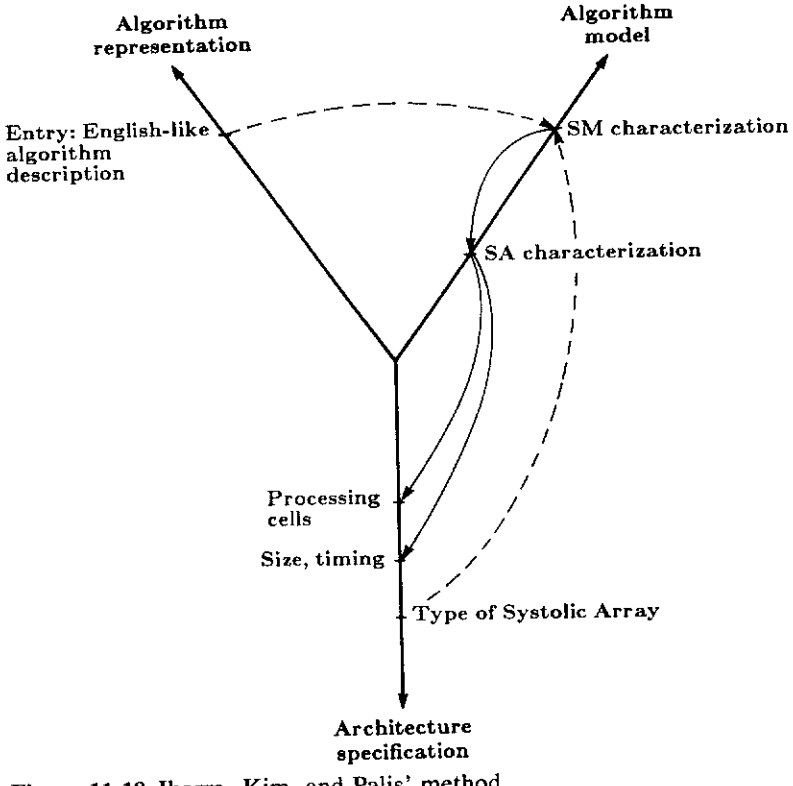


Figure 11.18 Ibarra, Kim, and Palis' method.

arrays, several general results on speedup, computational power, the effects of adding global control, and the use of one- and two-way communications were reported.

11.2.17. Leiserson, Rose, and Saxe's Method (81–84)

Description

This method starts with the design of a synchronous circuit (not necessarily systolic) whose correctness is either obvious or easily verifiable. This design is modeled as a finite, rooted, vertex-weighted, edge-weighted, directed multigraph, where nodes represent functional PEs and edges represent interconnections. Weights represent delays of nodes and register delays of interconnections. Transformations are then applied to the original design to obtain a systolic design without global broadcasts (Figure 11.19). The transformations applied include retiming, k-slowdown, broadcast and census elimination, coalescing, interlacing, code motion, resetting, register elimination, and parallel/serial compromises.

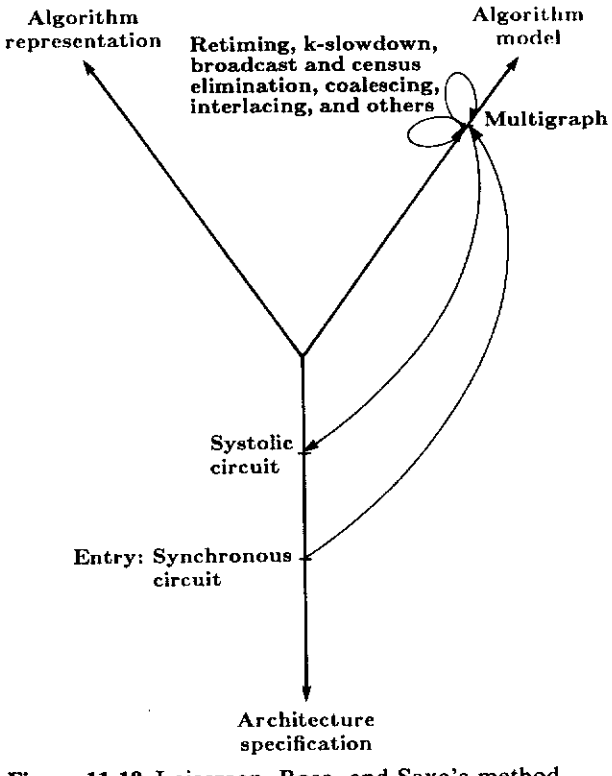


Figure 11.19 Leiserson, Rose, and Saxe's method.

Applicability
It applies to any synchronous system.

Capabilities
The function of PEs, layout, and number of pins are preserved by the transformations. Optimal retiming transformations can be selected either by reducing this problem to an efficiently solvable mixed-integer linear-programming problem such that the transformed circuit has the smallest clock period, or by solving a linear-programming dual of a minimum-cost flow problem such that the total number of registers is minimum. Extensions of the optimization procedures used can also take into account fanout, interconnection-bus width, multiple hosts, host timing constraints, and geometric constraints like the number of registers per interconnection.

Results
Systolic designs were derived from synchronous versions of a digital correlator and palindrome recognizer. Other derived circuits include priority

queues, search trees, priority multiqueue, counters, matrix–vector multiplication, matrix–matrix multiplication, and LU decomposition.

11.2.18. Chen and Mead's Method (19, 25)

Description
The goal is to verify that a given systolic design computes the function for which it was intended instead of the generation of a systolic architecture to compute a given function. However, one can see this method as the verification component of a design methodology in which systolic architectures are designed heuristically. Given a systolic architecture, the method generates a CRYSTAL program that describes the algorithm executed by the architecture as a set of space-time equations (19). This representation consists of several equations describing processes executed by local PEs, equations describing connections between PEs, functions representing data streams, and functions describing the relation between the structure of input and output data and the systolic-array structure. From fixed-point theory, the minimum solution of the system of recursive equations is the function computed by the systolic architecture (Figure 11.20).

Applicability
The generality and power of the formalism used makes the methodology widely applicable; however, for the same reason, it is not clear how practical and feasible it is to automate the steps and reasoning involved in this method.

Capabilities
Any systolic array with homogeneous or heterogeneous PEs and interconnections, and synchronous and self-timed systems can be verified.

Results
The method has been demonstrated in verifying the correctness of published designs for synchronous and self-timed systolic architectures for matrix–matrix multiplication.

11.2.19. Kuo, Levy, and Musicus' Method (20)

Description
This method starts from the knowledge of the action and position of each PE in the systolic array, the data "waves" present, their movements, and the way their components are indexed. A "wave" is simply a collection of related data that moves as a block during execution such that the relative positions are preserved (e.g., a matrix). By inspection, Space-Time-Data (STD) equations can be derived for each data wave. These equations relate

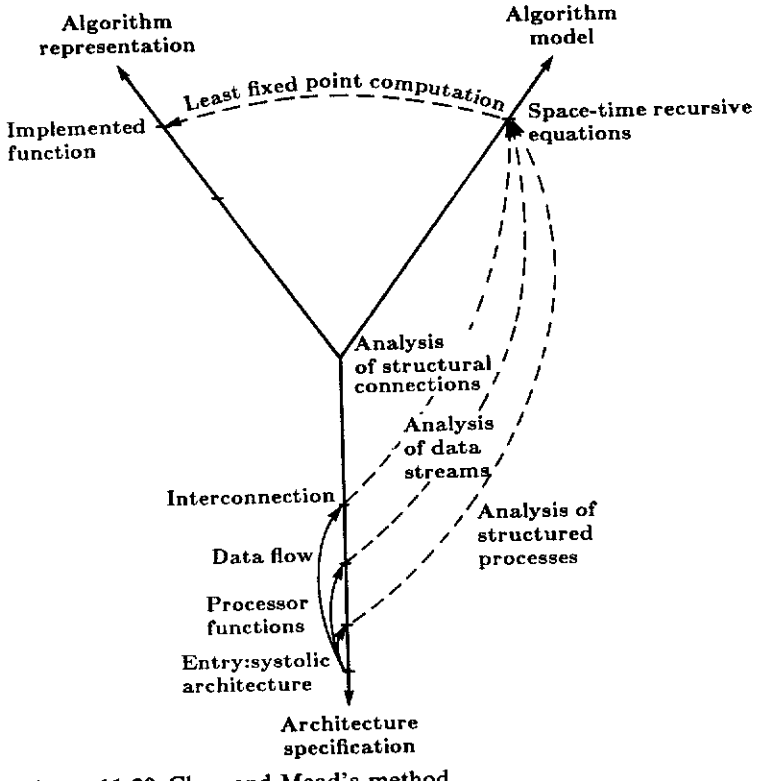


Figure 11.20 Chen and Mead's method.

the PE coordinates, time, and indices of data for each wave. Two functions can be derived from the STD equations: the position function and the memory function. The position function gives the coordinates of the PE at which some indexed data arrive at a given instant of time. The memory function is the inverse of the position function and gives the index of the data that arrives at a given PE at a given time. The direction and speed of data movement are described by velocity vectors that correspond to the difference between the coordinates of a PE receiving the data and those of the PE sending the same data. Verification is done by simulating the systolic network to either (a) track the activity of each PE over time by using the memory functions to identify the data being used at any given time, or (b) track each wave of data through the array by using the position functions to identify the PE being visited by a piece of data and the memory functions to identify other data present in that PE. If this simulation does exactly the same operations on the same data as the original algorithm, systolic algorithm is correct (Figure 11.21).

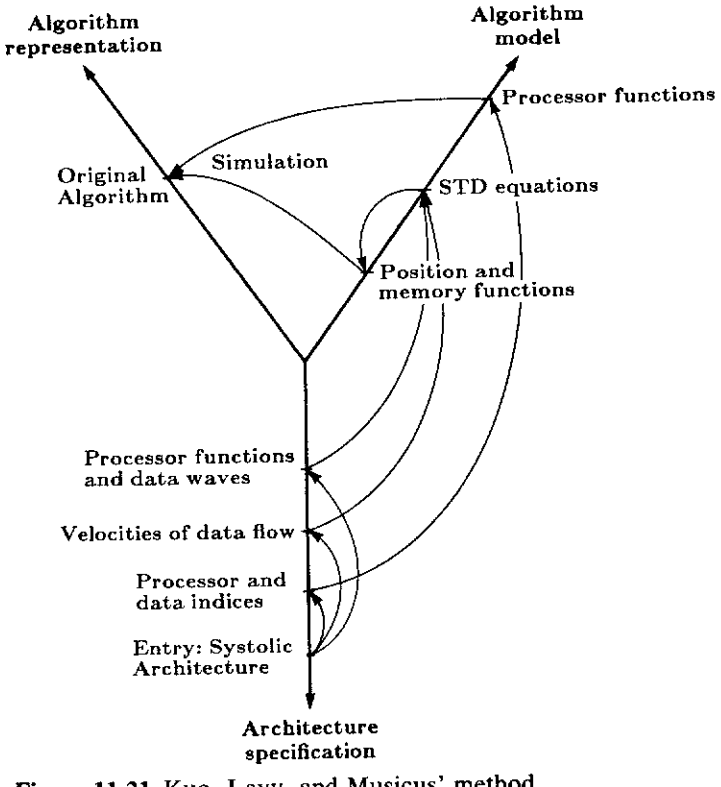


Figure 11.21 Kuo, Levy, and Musicus' method.

Applicability
This methodology is best suited for verifying spatially invariant systolic arrays for which data flows are independent of the values computed by the PEs. In practice, this means that the systolic algorithms must have regular dataflows through an array with regular geometry.

Capabilities
Computational wavefronts can be rigorously described by position and memory functions that are linear functions of time, data indices, and PE coordinates.

Results
The verification of matrix–matrix multiplication on a hexagonal array has been reported.

11.3. Final Remarks

In this chapter, we have surveyed 19 systematic methods for synthesizing algorithmically specified VLSI computational arrays. From a global point of view, it is clearly indicated that the two greatest limitations in the state of the art of existing transformational systems are the nonexistence of powerful systematic semantic transformations and the inability to systematically achieve optimality in the resulting designs. This will be the directions of future research in designing better methodologies.

Problems

1. Starting with an algorithmic description of matrix–matrix multiplication, derive the systolic array design of Figure 11.1 using the following methods: (a) Moldovan and Fortes' method, (b) Li and Wah's method, and (c) S. Y. Kung's method. (Hint—review references (75, 88, and 96.)
2. In "A mathematical model for the verification of systolic networks" by R. G. Mehlem and W. C. Rheinboldt (SIAM J. Comp., Vol. 13, No. 3, August 1984), a methodology is proposed for the verification of the correctness of systolic designs. Derive the Y-chart for this method and compare it with the verification methods mentioned in this chapter.
3. In reference (67), six systolic designs are described for the convolution problem. For each of the methodologies mentioned in this chapter, rederive those designs. (Review the relevant references for each methodology because most of them consider convolution as an example.)
4. (Project.) For each of the applications mentioned in Table 11.1, use the methods described in this chapter to verify, design, or redesign the systolic architectures described in the corresponding references. Compare the power, versatility, and effectiveness of the methods with respect to each application. List the main limitations and advantages of each method.

References

1. Agrawal DP, Pathak GC: Design of VLSI based multicomputer architecture for dynamic scene analysis. In Fu KS (ed): *VLSI for Pattern Recognition and Image Processing*, Springer-Verlag, New York, 1984.
2. Apostolico A, Negro A: Systolic algorithms for string manipulations. *Trans. on Computers*, Vol. C-33, No. 4, April 1984, pp. 361–364.
3. Banatre JP, Frison P, Quinton P: A network for the detection of words in continuous speech. *Acta Informatica*, Vol. 18, 1983, pp. 431–448.

4. Barbe DF: VHSIC systems and technology. *Computer,* Vol. 14, No. 2, Feb. 1981, pp. 13–22.
5. Barnwell TP III, Schwartz DA: Optimal implementations of flow graphs on synchronous multiprocessors. *Proc. 1983 Asilomar Conference on Circuits and Systems,* Pacific Grove, CA, Nov. 1983.
6. Baudet GM, Preparata FP, Vuillemin JE: Area-time optimal VLSI circuits for convolution. *Trans. on Computers,* Vol. C-32, No. 7, July 1983, pp. 684–688.
7. Bentley JL: A parallel algorithm for constructing minimum spanning trees. *Journal of Algorithms,* Jan. 1980, pp. 51–59.
8. Bentley JL, Kung HT: A tree machine for searching problems. *Proc. International Conference on Parallel Processing,* Aug. 1979, pp. 257–266.
9. Blackmer J, Kuekes P, Frank G: A 200 MOPS systolic processor. *Proc. SPIE: Real-Time Signal Processing IV,* Vol. 298, SPIE, 1981.
10. Bojanczyk A, Brent RP, Kung HT: Numerically stable solution of dense systems of linear equations using mesh-connected processors. *Journal on Scientific and Statistical Computing,* Vol. 5, No. 1, March 1984, pp. 95–104.
11. Bongiovanni G: A VLSI network for variable size FFTs. *Trans. on Computers,* Vol. C-32, No. 8, Aug. 1983, pp. 756–760.
12. Bongiovanni G: Two VLSI structures for the discrete Fourier transform. *Trans. on Computers,* Vol. C-32, No. 8, Aug. 1983, pp. 750–754.
13. Brent RP, Kung HT: Systolic VLSI arrays for linear-time GCD computation. In Anceau F, Aas EJ (eds): *VLSI* '83, North-Holland, Aug. 1983.
14. Brent RP, Luk FT, Loan CV: Computation of the singular value decomposition using mesh-connected processors, Technical Report 82-528, Cornell University, Ithaca, NY, March 1983.
15. Capello PR: VLSI Architecture for Digital Signal Processing. Ph.D. Thesis, Princeton University, Princeton, NJ, 1982.
16. Cappello PR, Steiglitz K: Unifying VLSI array designs with geometric transformations. *Proc. International Conference on Parallel Processing,* 1983, pp. 448–457.
17. Cappello PR, Steiglitz K: Unifying VLSI array design with linear transformations of space-time. In Preparata F (ed): *Advances in Computing Research,* JAI Press, Inc., 1984, pp. 23–65.
18. Cappello PR, Steiglitz K: Digital signal processing applications of systolic algorithms. In Kung HT, Sproull RR, Steele G Jr. (eds): *VLSI Systems and Applications,* Computer Science Press, Rockville, MD, 1981.
19. Chen M: Space-Time Algorithms: Semantics and Methodology. Technical Report 5090:TR:83, California Institute of Technology, May 1983.
20. Chen MC: A Methodology for Hierarchical simulation of VLSI Systems. Research Report, YALEU/DCS/RR-325, Yale University, New Haven, CT, Aug. 1984.
21. Chen MC: A Synthesis Method for Systolic Design. Research Report YALEU/DCS/RR-334, Yale University, New Haven, CT, Jan. 1985.
22. Chen MC: Synthesizing Systolic Designs. Research Report YALEU/DCS/RR-374, Yale University, New Haven, CT, March 1985.
23. Chen MC: The Generation of a Class of Multipliers: A Synthesis Approach to the Design of Highly Parallel Algorithms in VLSI. Research Report YALEU/DCS/RR-406, Yale University, New Haven, CT, July 1985.

24. Chen MC: A Parallel Language and its Compilation to Multiprocessor Machines or VLSI. Research Report YALEU/DCS/RR-412, Yale University, New Haven, CT, Aug. 1985.
25. Chen MC, Mead CA: Concurrent algorithms as space-time recursion equations. *Proc. USC Workshop on VLSI and Modern Signal Processing,* University of Southern California, Los Angeles, CA, Nov. 1982, pp. 31–52.
26. Cheng HD: Space-time domain expansion approach to VLSI and its application to pattern recognition and image processing. Ph.D. Thesis, Purdue University, West Lafayette, IN, 1985.
27. Cheng HD, and Fu KS: Algorithm partition for a fixed-size VLSI architecture using space-time domain expansion. *Proc. Seventh Symposium on Computer Arithmetic,* IEEE, June 1985.
28. Cheng HD, Fu KS: VLSI architectures for pattern matching using space-time domain expansion approach. *Proc. International Conference on Computer Design: VLSI in Computers,* IEEE 1985.
29. Cheng HD, Lin WC, Fu KS: *Proc. Seventh International Conference on Pattern Recognition,* IEEE, July 1984.
30. Cheng HD, Lin WC, Fu KS: Space-time domain expansion approach to VLSI and its application to hierarchical scene matching. *Trans. on Pattern Analysis and Machine Intelligence,* IEEE, May 1985, pp. 306–319.
31. Chiang YP, Fu KS: Parallel parsing algorithms and VLSI implementations for syntactic pattern recognition. *Trans. on Pattern Analysis and Machine Intelligence,* IEEE, Vol. PAMI-6, No. 5, May 1984, pp. 578–580.
32. Clarke MJ, Dyer CR: Systolic array for a dynamic programming application. *Proc. 12th Workshop on Applied Imagery Pattern Recognition,* IEEE, 1983.
33. Cohen D: Mathematical approach to iterative computation networks. *Proc. Fourth Symposium on Computer Arithmetic,* IEEE, 1978, pp. 226–238.
34. Cohen D, Tyree V: VLSI system for synthetic aperture radar (SAR) processing. In *Digital Processing of Aerial Images,* Vol. 186, SPIE, May 1979, pp. 166–177.
35. Dyer CR, Clarke MJ: Optimal curve detection in VLSI. *Proc. Conference on Computer Vision and Pattern Recognition,* IEEE, 1983, pp. 161–162.
36. Fisher AL: Systolic algorithms for running order statistics. In Kung HT, et al. (eds): *Signal and Image Processing VLSI Systems and Computations,* Computer Science Press, Rockville, MD, Oct. 1981, pp. 265–271.
37. Fortes JAB: Algorithm transformations for parallel processing and VLSI architecture design. Ph.D. Thesis, University of Southern California, Los Angeles, Dec. 1983.
38. Fortes JAB, Parisi-Presicce F: Optimal linear schedules for the parallel execution of algorithms. *Proc. International Conference on Parallel Processing,* IEEE, Aug. 1984.
39. Fortes JAB, Raghavendra CS: Dynamically reconfigurable fault-tolerant array processors. *Proc. 14th International Conference on Fault-Tolerant Computing,* IEEE, 1984.
40. Fortes JAB, Moldovan DI: Parallelism detection and transformation techniques useful for VLSI algorithms. *Journal of Parallel and Distributed Computing,* Vol. 2, Academic Press, 1985.
41. Fortes JAB, Moldovan DI: Data broadcasting in linearly scheduled array pro-

cessors. *Proc. 11th Annual Symposium on Computer Architecture*, ACM/IEEE, 1984, pp. 224–231.
42. Foster MJ, Kung HT: The design of special-purpose VLSI chips. *Computer*, Vol. 13, No. 1, Jan. 1980, pp. 26–40.
43. Foster MJ, Kung HT: Recognize regular languages with programmable building blocks. In *VLSI 81*, Academic Press, Aug. 1981, pp. 75–84.
44. Fouse SD, Nudd GR, Cumming AD: A VLSI architecture for pattern recognition using residue arithmetic. *Proc. Sixth International Conference on Pattern Recognition*, IEEE, 1982.
45. Frison P, Quinton P: A VLSI parallel machine for speech recognition. *Proc. ICASSP*, 1984, pp. 25B.3.1–25B.3.4.
46. Fu KS: VLSI for Pattern Recognition and Image Processing, Springer-Verlag, New York, 1984.
47. Gajski DD, Kuhn RH: Guest editor's introduction: New VLSI tools. *Computer*, Vol. 16, No. 12, Dec. 1983, pp. 11–14.
48. Gannon D: Pipelining array computations for MIMD parallelism: A functional specification. *Proc. International Conference on Parallel Processing*, 1982, pp. 284–286.
49. Gentleman WM, Kung HT: Matrix triangularization by systolic arrays. *Proc. SPIE: Real-Time Signal Processing IV*, Vol. 298, 1981.
50. Guibas LJ, Kung HT, Thompson CD: Direct VLSI implemention of combinatorial algorithms. *Proc. Conference Very Large Scale Integration*, California Institute of Technology, Jan. 1979, pp. 509–525.
51. Guibas LJ, Liang FM: Systolic stacks, queues, and counters. *Proc. Conference on Advanced Research in VLSI*, Massachusetts Institute of Technology, Jan. 1982.
52. Horowitz E: VLSI architecture for matrix computations. *Proc. International Conference on Parallel Processing*, Aug. 1979, pp. 124–127.
53. Hwang K, Cheng YH: VLSI computing structure for solving large-scale linear system of equations. *Proc. International Conference on Parallel Processing*, Aug. 1980, pp. 217–227.
54. Hwang K, Su SP: VLSI architectures for feature extraction and pattern classification. *International Journal on Computer Vision, Graphics, and Image Processing*, Vol. 24, Academic Press, Nov. 1983, pp. 215–228.
55. Ibarra O, Kim S, Palis M: Designing systolic algorithms using sequential machines. *Proc. 25th Annual Symposium on Foundations of Computer Science*, ACM, Oct. 1984, pp. 46–55.
56. Ibarra O, Kim S, Palis M: Some results concerning linear iterative (systolic) arrays. *Journal of Parallel and Distributed Computing*, Vol. 2, 1985, pp. 182–218.
57. Johnsson L, Cohen D: Mathematical approach to modeling the flow of data and control in computational Networks. In Kung HT, et al (eds): *VLSI Systems and Computations*, Computer Science Press, Rockville, MD, 1981, pp. 213–225.
58. Johnsson L, Cohen D: Computational arrays for the discrete Fourier transform. *Proc COMPCON*, 1981, pp. 236–244.
59. Johnsson L, Weiser U, Cohen D, Davis A: Towards a formal treatment of VLSI arrays. *Proc. Second Caltech Conference on VLSI*, California Institute of Technology, Jan. 1981.

60. Jover JM, Kailath T: Design framework for systolic-type arrays. *Proc. ICASSP*, 1984, pp. 8.5.1–8.5.4.
61. Jutand F, Demassieux N, Viard D, Chollet G: VLSI architectures for dynamic time warping using systolic arrays. *Proc. ICASSP*, pp. 34A.5.1–34A.5.4, IEEE, 1984.
62. Kuhn RH: Transforming algorithms for single-stage and VLSI architectures. *Proc. Workshop on Interconnection Networks for Parallel and Distributed Processing*, April 1980, pp. 11–19.
63. Kuhn RH: Optimization and interconnection complexity for parallel processors, single stage networks and decision trees, Ph.D. Thesis, Technical Report 80-1009, University of Illinois, Urbana-Champaign, IL, 1980.
64. Kung HT: Special-purpose devices for signal and image processing: An opportunity in VLSI. *Proc. SPIE: Real-Time Signal Processing III*, Vol. 241, July 1980, pp. 76–84.
65. Kung HT: Highly concurrent systems introduction to VLSI system. In Mead CA, Conway LA (eds): *Introduction to VLSI Systems*, Addison-Wesley, 1980.
66. Kung HT: Use of VLSI in algebraic computation: Some suggestions. *Proc. Symposium on Symbolic and Algebraic Computation*. ACM SIGSAM, Aug. 1981, pp. 218–222.
67. Kung HT: Why systolic architecture. *Computer*, Vol. 15, No. 1, Jan. 1982, pp. 37–46.
68. Kung HT, Lehman PL: Systolic (VLSI) arrays for relational database operations. *Proc. International Conference Management of Data*, ACM SIGMOD, May 1980, pp. 105–116.
69. Kung HT, Leiserson CE: Systolic arrays (for VLSI). In Sparse Matrix Proc. 1978, pp. 256–282.
70. Kung HT, Lin WT: An algebra for VLSI algorithm design. *Proc. Conference on Elliptic Problem Solvers*, Monterey, CA, 1983.
71. Kung HT, Picard RL: Hardware pipelines for multi-dimensional convolution and resampling. *Proc. Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, Nov. 1981, pp. 273–278.
72. Kung HT, Ruance LM, Yen DWL: A two-level pipelined systolic array for convolutions. In Kung HT, et al (eds): *VLSI Systems and Computations*, Computer Science Press, Rockville, MD, Oct. 1981, pp. 255–264.
73. Kung HT, Song SW: A systolic 2-D convolution chip, Technical Report CMU-CS-81-110, Carnegie-Mellon University, Pittsburgh, PA, March 1981.
74. Kung SY: From transversal filter to VLSI wavefront array. *Proc. Internationala Conference on VLSI*, IFIP, 1983.
75. Kung SY: On supercomputing with systolic/wavefront array processors. *Proc. IEEE*, Vol. 72, No. 7, 1984.
76. Kung SY, Hu YH: A highly concurrent algorithm and pipelined architecture for solving toeplitz systems. *Trans. on Acoustics, Speech, and Signal Processing*, Vol. ASSP-31, No. 1, Feb. 1983, pp. 66–76.
77. Kuo CJ, Levy BC, Musicus BR: The specification and verification of systolic wave algorithms. In *VLSI Signal Processing*, IEEE Press, 1984.
78. Lam M, Mostow J: A transformational model of VLSI systolic design. *IFIP Sixth International Symposium on Computer Hardware Descriptive Languages and their Applications*, Carnegie-Mellon University, Pittsburgh, PA, May 1983.

79. Lehman PL: A systolic (VLSI) array for processing simple relational queries. In Kung HT, Sproull RF, Steele GI (eds): *VLSI Systems and Computations*, Computer Science Press, Rockville, MD, 1981.
80. Leiserson CE: Systolic priority queues. *Proc. Conference on Very Large Scale Integration*, California Institute of Technology, Jan. 1979, pp. 199–214.
81. Leiserson CE: Systolic and semisystolic design (extended abstract). *Proc. International Conference on Computer Design/VLSI in Computers*, IEEE 1983.
82. Leiserson CE: Area-efficient VLSI Computations. Massachusetts Institute of Technology Press, Cambridge, MA, 1983.
83. Leiserson CE, Rose FM, Saxe JB: Optimizing synchronous circuitry by retiming. In Bryant R (ed): *Proc. Third Caltech Conference on Very Large Scale Integration*, Computer Science Press, Rockville, MD, 1983.
84. Leiserson CE, Saxe JB: Optimizing synchronous systems. *Twenty-second Annual Symposium on Foundations of Computer Science*, ACM, Oct. 1981, pp. 23–36.
85. Li GJ: Array pipelining algorithms and pipelined array processors. M.Sc. Thesis, Institute of Computer Technology, Chinese Academy of Science, Beijing, China, 1981.
86. Li GJ, Wah BW: Optimal design of systolic arrays for image processing. *Proc. Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, IEEE, Oct. 1983, pp. 134–141.
87. Li GJ, Wah BW: The design of optimal systolic algorithms. *Proc. Computer Software and Applications Conference*, IEEE, 1983, pp. 310–319.
88. Li GJ, Wah BW: The design of optimal systolic arrays. *Trans. on Computers*, Vol. C-34, No. 10, Jan. 1985, pp. 66–77.
89. Liu HH, Hu KS: VLSI algorithm for minimum distance classification. *Proc. International Conference on Computer Design*, IEEE, 1983.
90. Liu HH, Fu KS: A VLSI systolic processor for fast seismic signal classification. *Proc. International Symposium on VLSI Technology, Systems and Applications*, Taipei, Taiwan, March 30–April 1, 1983.
91. Liu PS, Young TY: VLSI array architecture for picture processing. In Fu KS, Kunii TL (eds): *Picture Engineering*, Springer Verlag, New York, Vol. 6, 1982.
92. McCanny JV, McWhirter JG: Implementation of signal processing functions using 1-bit systolic arrays. *Electronic Letters*, Vol. 18, No. 6, March 1982, pp. 241–243.
93. McCanny JV, McWhirter JG: Completely iterative, pipelined multiplier array suitable for VLSI. *Proc. IEEE*, Vol. 129, No. 2, April 1982, pp. 40–46.
94. McWhirter JG: Systolic array for recursive least-square minimization. *Electronics Letters*, Vol. 19, No. 18, Sept. 1983, pp. 729–730.
95. Miranker WL, Winkler A: Space-time representations of computational structures. *Computing*, Vol. 32, 1984, pp. 93–114.
96. Moldovan DI: On the analysis and synthesis of VLSI algorithms. *Trans. on Computers*, Vol. C-31, No. 11, Nov. 1982, pp. 1121–1126.
97. Moldovan DI: On the design of algorithms for VLSI systolic arrays. *Proc. IEEE*, Vol. 71, No. 1, Jan. 1983, pp. 113–120.
98. Moldovan DI, Fortes JAB: Partitioning of algorithms for fixed size VLSI architectures. Technical Report PPP-83-5, Department of Electrical Engineering Systems, University of Southern California, Los Angeles, CA, 1983.

99. Moldovan DI, Varma A: Design of algorithmically specialized VLSI devices. *Proc. International Conference on Computer Design: VLSI in Computers*, 1983, pp. 88–91.
100. Moldovan DI, Wu CI, Fortes JAB: Mapping an arbitrarily large QR algorithm into a fixed size VLSI array. *Proc. International Conference on Parallel Processing*, Aug. 1984.
101. Mostow J, Lam M: Transformational VLSI design: A progress report. Unpublished manuscript.
102. Narayan SS, Nash JG, Nudd GR: VLSI processor array for adaptive radar applications. *Proc. of SPIE 27th International Technical Symposium*, 1983.
103. Ni LM, Jain AK: Design of a pattern cluster using two-level pipelined systolic array. In Fu KS (ed): *VLSI for Pattern Recognition and Image Processing*, Springer-Verlag, New York, 1984.
104. Nishida S: Application of mapping algorithm to VLSI architectures, Technical Report, Department of Engineering Systems, University of Southern California, Los Angeles, 1984.
105. Ottmann T, Rosenberg AL, Stockmeyer LJ: A dictionary machine (for VLSI). Technical Report RC9060, IBM T. J. Watson Research Center, Yorktown Heights, NY, 1981.
106. Quinton P: The systematic design of systolic arrays. Technical Report 193, IRISA, April 1983.
107. Quinton P: Automatic synthesis of systolic arrays from uniform recurrent equations. *Proc. 11th Annual Symposium on Computer Architecture*, ACM/IEEE, 1984, pp. 208–214.
108. Ramakrishnan IV, Fussell DS, Sillberschatz A: On mapping homogeneous graphs on a linear array-processor model. *Proc. International Conference on Parallel Processing*. IEEE, 1983, pp. 440–447.
109. Savage C: A systolic data structure chip for connectivity problems. In Kung HT, Sproull RF, Steele GL Jr. (eds): *VLSI Systems and Computations*, Computer Science Press, Rockville, MD, 1981.
110. Schwartz DA, Barnwell TP III: A graph theoretic technique for the generation of systolic implementations for shift-invariant flow graphs. *Proc. ICASSP, IEEE*, 1984, pp. 8.3.1–8.3.4.
111. Song SW: On a high-performance VLSI solution to database problems. Ph.D. Dissertation Carnegie-Mellon University, Pittsburgh, PA, July 1981.
112. Stroll Z, Kang SC: VLSI-based image resampling for electronic publishing. In Fu KS (ed): *VLSI for Pattern Recognition and Image Processing*, Springer-Verlag, New York, 1984.
113. Travassos RT: Real-time implementation of systolic Kalman filters. Technical Report, Systolic Systems Inc., 1983.
114. Treleaven PC: VLSI processor architectures. *Computer*, Vol. 15, No. 6, June 1982, pp. 33–45.
115. Tur M, Goodman JW, Moslehi B, et al: Fiber-optic signal processor with applications to matrix-vector multiplication and lattice filtering. *Optics Letters*, Vol. 7, No. 9, Sept. 1982, pp. 463–465.
116. Ullman JD: Computational Aspects of VLSI, Computer Science Press, Rockville, MD, 1984.
117. Varman PJ, Fussell DS, Ramakrishnan IV, Sillberschatz A: Robust systolic

algorithms for relational database operations. *Proc. Symposium on Real Time Systems*, Dec. 1983.

118. Varman PJ, Ramakrishnan IV: Dynamic programming and transitive closure on linear pipelines. *Proc. International Conference on Parallel Processing*, 1984, pp. 359–364.
119. Weiser V, Davis A: A wavefront notion tool for VLSI array design. In Kung HT, et al (eds): *VLSI Systems and Computations*, Computer Science Press, Rockville, MD, 1981.
120. Weste V, Burr DJ, Ackland BD: Dynamic time warp pattern matching using an integrated multiprocessing array. *Trans. On Computers*, Vol. C-32, No. 8, Aug. 1983, pp. 731–744.
121. Wing O: A content-addressable systolic array for sparse matrix computation. Technical Report, Columbia University, New York, 1983.
122. Yeh CS, Reed IS, Truong TK: Systolic multipliers for finite fields. *Trans. on Computers*, Vol. C-33, No. 4, April 1984, pp. 357–360.
123. Yen DWL, Kulkarni AV: Systolic processing and an implementation for signal and image processing. *Trans. on Computers*, Vol. C-31, No. 10, Oct. 1982, pp. 1000–1008.
124. Zhang CN, Yun DYY: An area-time optimal systolic network for discrete Fourier transform. *Proc. 11th Annual International Symposium on Computer Architecture*, ACM/IEEE, June 1984.