SYSTEMATIC APPROACHES TO THE DESIGN OF ALGORITHMICALLY SPECIFIED SYSTOLIC ARRAYS

J.A.B. Fortes, K.S. Fu and B.W. Wah**

School of Electrical Engineering
Purdue University
West Lafayette, IN 47907

1. INTRODUCTION

Evolution in VLSI technology allows many algorithms in pattern recognition and image and signal processing to be implemented on a VLSI chip using multiple, regularly-connected processing elements (PEs) that exploits the great potential of pipelining and multiprocessing. This type of array processor has been referred to as a *systolic array*.

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 requirements of the design are satisfied. In this paper we survey seventeen methodologies proposed in the literature. The applicability, capabilities and results derived for each methodology are identified.

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 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 representations. A transformational system as a three dimensional space where dimensions (or axes) are associated with algorithm representation, algorithm model and architecture specification. To the axis of algorithm representation, we associate different forms or levels in which an algorithm can be presented 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 (Fig. 1) 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.** Arcs drawn in full lines represent systematic transformations whereas those drawn in broken lines represent ad-hoc transformations. These Y charts will allow us to classify and describe the large number of approaches taken for systematic systolic-array design. As an example, we use Fig. 2 to explain Kuhn's approach [Kuh80a].

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. The following step consists of, in an ad-hoc manner, introducing additional subscripts for variable referencing so 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 (Fig. 2). 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 modes). 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 similar to those used in optimizing compilers. A reindexing 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 identifying the iterations where input variables are used, the size, dimension and input/output ports of the architecture can be systematically generated. To each point in the projected space, it corresponds a processor 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 is resulted from applying the reindexing transformation. This completes our example of the use of Y-charts to explain a methodology.

In the following sections we will describe the seventeen 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.

(a) Cohen, Johnsson, Weiser and Davis' method [Coh78, JoC81a, JoC81b, JoW81, WeD81]:

Description: Starting from a mathematical expression involving subscripted variables, which conceptually represent data sequenced in time or space, this method starts 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., allocation of a data stream to processors). 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 ordering of execution of the operations can be derived according to known operation precedence rules. The number, placement and interconnection of operator cells can also be derived. Timing and storage requirements are inferred from the placement of delay cells (which correspond to the Z operator) (Fig. 3).

Applicability: This method is best applicable to algorithms that can be described by relatively simple and concise mathematical expressions. *Capabilities*: Computational rate, performance, delay, modularity and size can be derived from the equations; interconnection and communication characteristics can also be derived when the architecture is regular; search for the optimal design is done in ad hoc manner.

Results: Formal derivations have been reported for architectures intended for the following problems: Finite Impulse Response (FIR) filters, matrix-vector product, string matching, solution of triangular linear equations, product of band matrices and multiplication and division of polynomials. A set of data-set operators defined in terms of the operator Z was also proposed for treating sets of data as wavefront entities in expressions and their graphical representation.

(b) Mostow and Lam's method (SYS) [LaM83, MoL83]:

Description: SYS accepts as input an algorithm suitable for systolic implementation, i.e., an algorithm obtained from a high-level specification by software transformations that result in segments of code executed repeatedly with a regular pattern of data access. The algorithm is then mapped into a systolic design described by a structure and a driver. The structure describes the hardware cells (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 (Fig. 4).

Applicability: SYS can process algorithms with simple FOR-loops and BEGIN-END blocks, simple unnested function calls, 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. The specification of structure includes the number and dimensionality of ports of cells, hierarchical definition of cells, arrays and compounds of cells, and interconnections among them including broadcasts and directional links. The driver describes data streams and timing schemes including 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 non-systolic designs using a transformational system related to SYS include a chip for color shading and hidden-surface elimination and a multi-chip switching network for marker passing semantic networks.

(c) Gannon's method [Gan82]:

Description: From an algorithm, a functional specification is derived by using vector operators where parallelism is explicitly represented. 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 processing cells of the systolic architecture. The vector operators include a product operator which represents the concurrent operation of basic functions, permutation and data-movement operators, chain operator which represents the iterated composition of basic functions, and the systolic-iteration operator which describes operations where basic functions are "reused." The global functional specification of the algorithm is viewed as a data-flow graph which, depending on the properties of the function and operators used, can be mapped into a systolic architecture. Different architectures result by expressing the same algorithm with different operators (Fig. 5).

Applicability: This method seems to be suited to those algorithms which can be reexpressed by using vector operators.

Capabilities: The functional description of cells and the interconnection topology can be easily derived; additional information like data movement, timing, etc., is present only in implicit form.

Results: Then design of a recurrence solver was derived. The formalism used proved the theoretical result that systolic versions of computation graphs perform asymptotically as fast as fully concurrent execution of the original data-flow graph.

(d) H. T. Kung and Lin's method [KuL83]:

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 matricial expressions of the form (a) $v \leftarrow Av + bx$ and (b) $y = c^T v$. In these expressions, x represents the input, y represents the output and v represents variables generated by implicit functions; the $(n \times n)$ matrix A and the

* Research supported by National Science Foundation Grant ECS 80-16580.

** We borrowed from [GaK83] 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.

column vectors b and c represent the delay cycles between the availability and the use of variables, and each entry is either zero or Z^{-k}, where k corresponds to the number of delays. For example, the i'th component of v in Expression (a) is $v_i \leftarrow Z^{-a_{i1}}v_1 + \ldots + Z^{-a_{in}}v_n + Z^{-b_i}x$, and it means that $v_i(t) = f_i[\, v_1(t-a_{i1}), \ldots, v_n(t-a_{in}), x(t-b_i)]$ for some implicit function f_i associated with node v_i. To the canonical representation, algebraic transformations are then applied. There are two main types of transformations, namely, retiming transformations and "k-slowing" transformations 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 from algebraic representations and a hardware related representation denoted as a 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 (Fig. 6).

Applicability: Suitable for algorithms for which a canonical algebraic representation can be found.

Capabilities: Functional description of cells, interconnections, timing for input/output and data communication can be derived systematically; designs and transformations can be expressed algebraically without drawings; theoretical results on retiming and k-slowed designs can be easily proved using 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 architectures for LU decomposition.

(e) Kuhn's method [Kuh80a, Kuh80b]:

Description: As described early in this paper.

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

Capabilities: Size, dimension, topology, input-output ports, execution time, data movement, timing and functional descriptions of cells 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 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.

(f) Moldovan and Fortes' method [Mol82, Mol83, MoF83, MoV83, For83, FoM84, FoM85, FoP84, FoR84a, FoR84b, MoW84, Nis84]:

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, each computation has an index rather than associating the body of loops with the corresponding loop indices. The algebraic representation of the algorithm is then transformed by local and global transformations. Local transformations are used to rewrite computations which are then mapped into the functional and structural specification of the cells of the systolic architecture. Global transformations are composed of time and space transformations and are used to restructure the algorithm. They are chosen so that the new algorithm has a set of dependencies which 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 (Fig. 7).

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

Capabilities: Because this method is an extension of Kuhn's approach, it has the same capabilities as that method. Additionally, it allows to use or eliminate broadcasting, to design fixed size architectures for arbitrarily large algorithms, to implement fault tolerance schemes and to optimize 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 decomposition 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.

(g) Miranker and Winkler's method [MiW84]:

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 or mathematical expressions by using the properties of the operators in an ad-hoc fashion. The other extension is to use 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 (Fig. 8).

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

Capabilities: Size, dimension, topology, data movement and timing, functional description of cells 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.

(h) Cappello and Steiglitz's method [CaS83a, CaS83b]:

Description: Starting with 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 cells 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 data flow are derived systematically. By selecting different geometric transformations, distinct geometric representations and their corresponding architectures can be derived (Fig. 9).

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 represented as matrix transformations. 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 asymptotically optimal, and alternative designs were provided.

(i) S. Y. Kung's method [KuS83, KuS84]:

Description: Given a Signal Flow Graph (SFG) representing an algorithm, this method starts by choosing basic operational modules which correspond to the functional description of cells of the architecture. Then, localization rules are 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 so that at least one unit of time is allowed for communication of a signal between two nodes. Then, 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 cells and edges into interconnections. Timing and data movements are derived from the basic modules because of the localized spatial and temporal characteristics of the SFG (Fig. 10).

Applicability: To all algorithms described by computable SFGs with some regularity.

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

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

(j) Quinton's method [Qui83, Qui84]:

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, and it projects D into space along a preselected direction which is chosen so 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 (they are quasi-affine functions) are known, the systolic array can be systematically generated from D. The procedure maps each point of D into a cell which computes the recurrence function, and receives and sends data from and to cells which are the image of points dependent and depending on the point under consideration with delays given by the timing function (Fig. 11).

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

Capabilities: The functional description of cells, 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 allows the derivation of arrays for LU decomposition and dynamic programming.

(k) Ramakrishnan, Fussell and Silberschatz's method [RaF83, VaR84]:

Description: This method starts with a data-flow description of the algorithm (an acyclic program graph) which is partitioned into sets of vertices that are mapped into the same processor (this partitioning is called diagonalization). Then, a syntactically correct mapping is used to map computation vertices onto processors and time steps and the labels and edges to communication delays and interconnections (Fig. 12).

Applicability: The method applies only to homogeneous graphs with connected subgraphs satisfying certain properties; moreover, the method can only be used to generate linear arrays.

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

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

(l) Li and Wah's method [Li81, LiW83a, LiW83b, LiW84]:

Description: Starting with an algorithm described as a set of linear recurrence equations, this method derives three classes of parameters: velocities of data flows, spatial distributions of data and periods of data. The relationships among these parameters are represented as constraint vector equations which must be satisfied in order to obtain 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 processors. Optimal designs are then searched in the space of solutions satisfying the constraint equations. This search is done by ordered enumeration over a limited search space in time polynomial to the problem size. From the definition of the recurrence equations, the functional description of the cells is derived. From the parameters mentioned above, the interconnections among cells are found (Fig. 13).

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

Capabilities: Functional description of cells, timing and spatial distribution of data flows, execution time, number of processors 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 Transforms, polynomial multiplication, deconvolution, triangular matrix inversion and tuple comparison.

(m) Cheng and Fu's method [ChF83]:

Description: Starting with a recursive formula with several indices or a program-loop with a simple expression, this method starts by designing the basic computational cell to compute the simple expression. Then, this basic cell is expanded in time and space so 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 expansions 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 data-flow timing and direction as well as cell interconnections (Fig. 14). The method can be implemented at the gate level, register level, processing-unit level and system level. Since 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 to partition any problem suitable for the method and implement it on a fixed-size VLSI architecture.

Applicability: The method is suited to algorithms described by recurrence expressions or programs with loops.

Capabilities: Functional description of cells, timing and directions of data flows, interconnections, execution time and number of processors of the architecture can be systematically generated.

Results: This method has been applied to construct computational structures for (partitioned) vector inner product, (partitioned) matrix multiplication, convolution, (partitioned) comparison operations in relational databases, Fast Fourier Transform, hierarchical scene matching, partitioned transitive closure computation and partitioned recognition of context-free languages (dynamic programming).

(n) Jover and Kailath's method [JoK84]:

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 one given value was computed or as a stream of values in different stages of a computation. Due to 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) (Fig. 15).

Applicability: Suitable for algorithms from which one can easily identify Lines of Computation.

Capabilities: The topology of the systolic array can be derived from LOCs; additionally, throughput, efficiency, data interval, initial conditions, interval and external delays and pipelinability can be found from LOCs and the knowledge of execution times of basic operations.

Results: Three designs for matrix multiplication were derived.

(o) Schwartz and Barnwell's method [BaS83, ScB84]:

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 processors 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 so that correctness is preserved and data transfers can be simultaneous. The transformations of flow graphs used to achieve this consist of data interleaving and the cut-set or delay transformation that are shown to preserve equivalence. The transformed flow graph is mapped into a systolic array by mapping nodes into processors and delays and edges into interconnections (Fig. 16).

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 processors, interconnections, data movement and time. The optimality of the resulting designs can be analyzed.

Results: Architectures have been derived for FIR and IIR filters and the two multiplier Markel-Gray lattice filter.

(p) Leiserson, Rose and Saxe's method [Lei81, LeS81, LeR83]:

Description: This method starts with the design of a synchronous circuit (not necessarily systolic) whose correctness is obvious or easily verifiable. This design is modeled as a finite, rooted, vertex-weighted, edge-weighted, directed multigraph where nodes represent functional cells and edges represent interconnections. Weights represent delays of nodes and register delays of interconnections. Retiming and k-slowdown transformations are then applied to the original design in order to obtain a systolic design without global broadcasts (Fig. 17).

Applicability: It applies to any synchronous system.

Capabilities: The function of cells, layout and number of pins are preserved by the transformations. Optimal retiming transformations can be selected so that the transformed circuit has the smallest clock period by reducing this problem to an efficiently solvable mixed-integer linear programming problem. Optimal retiming transformations can also be selected so that the total number of registers is minimized by solving a linear-programming dual of a minimum-cost flow problem. Extensions of the optimization procedures used can also take into account fan-out, 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.

(q) Chen and Mead's method [ChM82, Che83]:

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 where systolic architectures are designed heuristically. Given a systolic architecture, the method generates a CRYSTAL program which describes the algorithm executed by the architecture as a set of space-time equations [Che83]. This representation consists of several equations describing processes executed by local cells, equations describing connections between cells, functions representing data streams, and functions describing the relation between the structure of input and output data and the systolic-array structure. The minimum solution of the system of recursion equations is (from fixed-point theory) the function computed by the systolic architecture (Fig. 18).

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 processing cells 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.

3. FINAL REMARKS

In this paper we have surveyed seventeen systematic methods for synthesizing algorithmically specified VLSI computational arrays. From a global point of view, it is clearly indicated that the two largest limitations in the state of the art of existing transformational systems are the inexistence 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.

REFERENCES

[BaS83] T.P. Barnwell III and D.A. Schwartz, "Optimal Implementations of Flow Graphs on Synchronous Multiprocessors," *1983 Asilomar Conf Circuits and Systems*, Nov. 1983.

[CaS83a] P.R. Cappello and K. Steiglitz, "Unifying VLSI Array Designs with Geometric Transformations," *1983 Conf on Parallel Processing*, pp. 448-457.

[CaS83b] P.R. Cappello and K. Steiglitz, *Unifying VLSI Array Design with Linear Transformations of Space Time*, Tech. Rep. TR-CS-83-08, Dept. of Computer Science, Univ. of Calif., Santa Barbara, 1983.

[ChM82] M.C. Chen and C.A. Mead, "Concurrent Algorithms as Space-Time Recursion Equations," *USC Workshop on VLSI and Modern Signal Processing*, pp. 31-52, Nov. 1982.

[Che83] M. Chen, *Space-Time Algorithms: Semantics and Methodology*, Tech. Rep. 5090:TR:83, Caltech, May 1983.

[ChF83] H. D. Cheng, W. C. Lin and K. S. Fu, *Space Time Domain Expansion Approach to VLSI and its Application to Hierarchical Scene Matching*, Tech. Rep., TR-EE-83-29, Purdue University, July 1983; Summary published *7th Conf on Pattern Recognition*, Montreal, Canada, July 1984.

[Coh78] D. Cohen, "Mathematical Approach to Iterative Computation Networks," *4th Symp on Computer Arithmetic*, 1978.

[For83] J. A. B. Fortes, *Algorithm Transformations for Parallel Processing and VLSI Architecture Design*, Ph.D. Thesis, Univ. of Southern California, Dec. 1983.

[FoM84] J.A.B. Fortes and D.I. Moldovan, "Data Broadcasting in Linearly Scheduled Array Processors," *11'th Symp on Comp Architecture*, 1984.

[FoM85] J.A.B. Fortes and D.I. Moldovan, "Parallelism Detection and Transformation Techniques Useful For VLSI Algorithms," *J of Parallel and Distributed Computing*, 1985.

[FoP84] J.A.B. Fortes and F. Parisi-Presicce, "Optimal Linear Schedules for the Parallel Execution of Algorithms," *1984 Conf on Parallel Processing*, Aug. 1984.

[FoR84a] J.A.B. Fortes and C.S. Raghavendra, "Dynamically Reconfigurable Fault Tolerant Array Processor," *14'th Conf on Fault Tolerant Computing*, 1984.

[FoR84b] J.A.B. Fortes and C.S. Raghavendra, *Gracefully Degradable Array Processors*, Tech. Rep. TR-EE-84-15, Sch. of Electrical Eng., Purdue Univ., June 1984.

[GaK83] D.D. Gajski and R.H. Kuhn, "Guest Editor's Introduction: New Tools," *IEEE Computer*, No. 12, Dec. 1983.

[Gan82] D. Gannon, *Pipelining Array Computations for MIMD Parallelism: A Functional Specification*, *1982 Conf on Parallel Processing*, 1982.

[JoC81a] L. Johnsson and D. Cohen, "A Mathematical Approach to Modeling the Flow of Data and Control in Computational Networks," in *VLSI Systems and Computations*, H. T. Kung et al. eds, Comp. Sci. Press, Oct. 1981.

[JoC81b] L. Johnsson and D. Cohen, "Computational Arrays for the Discrete Fourier Transform," *COMPCON 81*.

[JoW81] L. Johnsson, U. Weiser, D. Cohen and A. Davis, "Towards a Formal Treatment of VLSI Arrays," *Second Caltech Conf on VLSI*, Jan. 1981.

[JoK84] J.M. Jover and T. Kailath, "Design Framework for Systolic-Type Arrays," *Proc of ICASSP 84*, 1984.

[Kuh80a] R.H. Kuhn, *Optimization and Interconnection Complexity for Parallel Processors, Single Stage Networks and Decision Trees*, Ph.D. Thesis, Dept. of Comp. Sci., Rpt. 80-1009, Univ. of Illinois, Urbana-Champaign, Illinois 1980.

[Kuh80b] R.H. Kuhn, "Transforming Algorithms for Single-Stage and VLSI Architectures," *Workshop on Interconnection Networks for Parallel and Distributed Processing*, 1980.

[KuL83] H.T. Kung and W.T. Lin, "An Algebra for VLSI Algorithm Design," *Conf on Elliptic Problem Solvers*, 1983.

[KuS83] S. Y. Kung, "From Transversal Filter to VLSI Wavefront Array," *Proc Int'l Conf on VLSI*, IFIP, 1983.

[KuS84] S.Y. Kung, "On Supercomputing with Systolic/Wavefront Array Processors," *Proc IEEE*, Vol. 72, No. 7, 1984.

[LaM85] M. Lam and J. Mostow, "A Transformational Model of VLSI Systolic Design," *IFIP 6th Symp Computer Hardware Descriptive Languages and Applications*, 1983.

[Lei81] C. E. Leiserson, *Area-Efficient VLSI Computations*, Ph.D. Thesis Dept. of Comp. Sci., Carnegie-Mellon Univ., 1981.

[LeS81] C. E. Leiserson and J. B. Saxe, "Optimizing Synchronous Systems," *22nd Symp Found Computer Science*, 1981.

[LeR83] C.E. Leiserson, F.M. Rose and J.B. Saxe, "Optimizing Synchronous Circuitry by Retiming," *Proc Third Caltech Conference on VLSI*, ed. R. Bryant, Comp. Sci. Press, 1983.

[Li81] G.J. Li, *Array Pipelining Algorithms and Pipelined Array Processors*, M.Sc thesis, Institute of Comput. Tech., Chinese Academy of Science, 1981.

[LiW83a] G.J. Li and B. W. Wah, "The Design of Optimal Systolic Algorithms," *Proc of Computer Software and Applications Conference*, pp. 310-319, 1983.

[LiW83b] G.J. Li and B.W. Wah, "Optimal Design of Systolic Arrays for Image Processing," *Proc of Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, pp. 134-141, Oct. 1983.

[LiW84] G.J. Li and B.W. Wah, "The Design of Optimal Systolic Arrays," *IEEE Trans on Computers*, Vol. C-33, No. 10, Oct. 1984.

[MiW84] W.L. Miranker and A. Winkler, "Space-Time Representations of Computational Structures," *Computing* Vol. 32, pp. 93-114, 1984.

[Mol82] D. I. Moldovan, "On the Analysis and Synthesis of VLSI Algorithms," *IEEE Trans on Computers*, Vol. C-31, No. 11, pp. 1121-1126, Nov. 1982.

[Mol83] D.I. Moldovan, "On the Design of Algorithms for VLSI Systolic Arrays," *Proc of the IEEE*, Vol. 71, No. 1, pp. 113-120, Jan. 1983.

[MoF83] D.I. Moldovan and J.A.B. Fortes, *Partitioning of Algorithms for Fixed Size VLSI Architectures*, Tech. Rep. PPP-83-5, Dept. of Elec. Eng., USC, 1983.

[MoV85] D.I. Moldovan and A. Varma, "Design of Algorithmically Specialized VLSI Devices," *Proc 1985 Int'l Conf Comp Design VLSI in Computers*, 1985.

[MoW84] D.I. Moldovan, C.I. Wu and J.A.B. Fortes, "Mapping an Arbitrarily Large QR Algorithm into a Fixed Size VLSI Array," *Proc of 1984 Int'l Conf on Parallel Processing*, Aug. 1984.

[MoL83] J. Mostow and M. Lam, *Transformational VLSI Design: A Progress Report*, unpublished manuscript.

[Nis84] S. Nishida, *Application of Mapping Algorithm to VLSI Architectures*, Dept. of Electrical Engineering Systems, Univ. of Southern California, 1984.

[Qui83] P. Quinton, *The Systematic Design of Systolic Arrays*, Tech. Report 193, IRISA, April, 1983.

[Qui84] P. Quinton, "Automatic Synthesis of Systolic Arrays from Uniform Recurrent Equations," *Proc 11'th Ann Symp Comp Architecture*, 1984.

[RaF83] I. V. Ramakrishnan, D. S. Fussell and A. Silberschatz, "On Mapping Homogeneous Graphs on a Linear Array-Processor Model," *Proc of 1983 Int'l Conf on Parallel Processing*, pp. 440-447, 1983.

[ScB84] D.A. Schwartz and T.P. Barnwell III, "A Graph Theoretic Technique for the Generation of Systolic Implementations for Shift-Invariant Flow Graphs," *Proc of ICASSP 84*, pp. 8.3.1-8.3.4, 1984.

[VaR84] P.J. Varman and I.V. Ramakrishnan, "Dynamic Programming and Transitive Closure on Linear Pipelines," *Int'l Conf Parallel Processing*, 1984.

[WeD81] V. Weiser and A. Davis, "A Wavefront Notion Tool for VLSI Array Design," in *VLSI Systems and Computations*, H. T. Kung et al. eds, Computer Science Press, Oct. 1981, pp. 226-234.

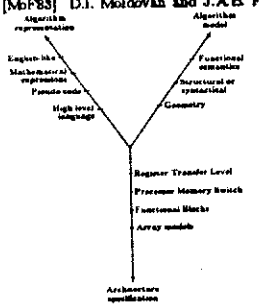

Fig. 1. Y-chart for transformation systems

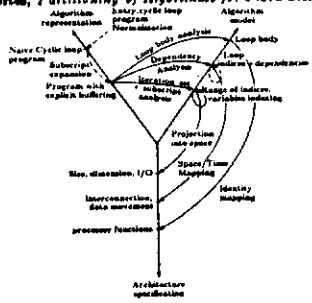

Fig. 2. Kuhn's method

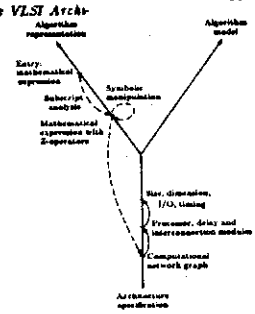

Fig. 3. Cohen, Johnson, Weiser and Davis' method

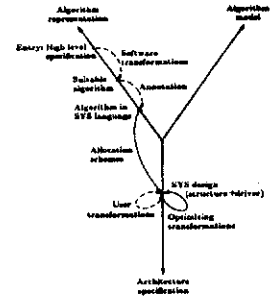

Fig. 4. Mostow and Lam's method

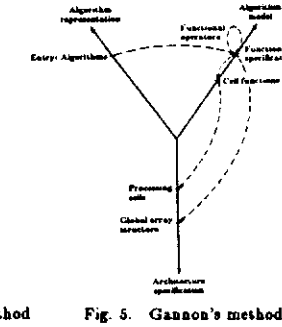

Fig. 5. Gannon's method

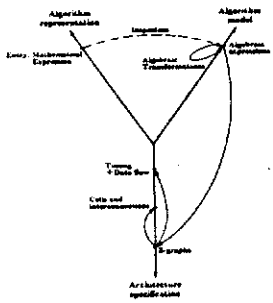

Fig. 6. Kung and Lin's method

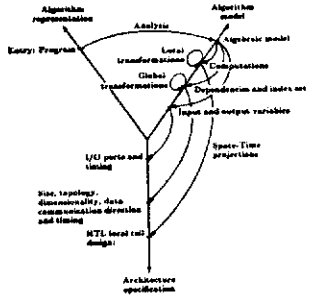

Fig. 7. Moldovan and Fortes' method

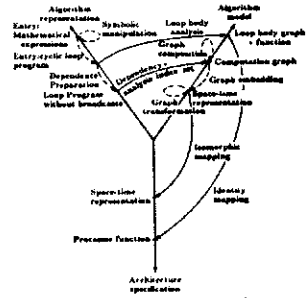

Fig. 8. Miranker and Winkler's method

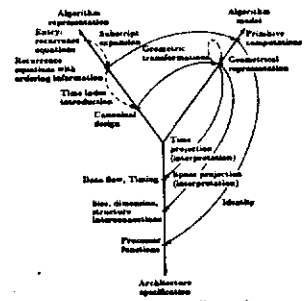

Fig. 9. Cappello and Steiglitz's method

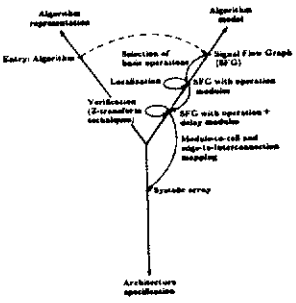

Fig. 10. S. Y. Kung's method

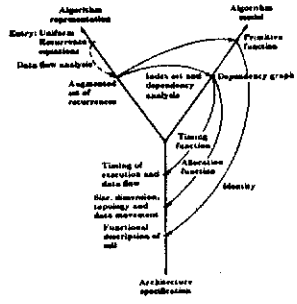

Fig. 11. Quinton's method

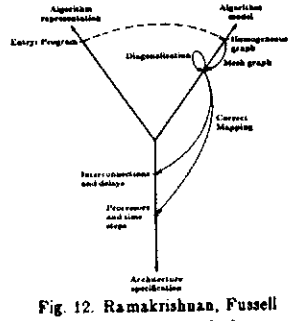

Fig. 12. Ramakrishnan, Fussell and Silberschatz's method

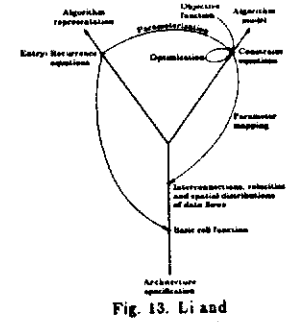

Fig. 13. Li and Wah's method

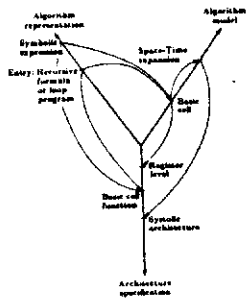

Fig. 14. Cheng and Fu's method

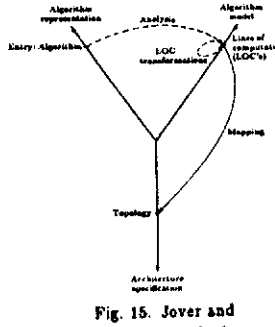

Fig. 15. Jover and Kailath's method

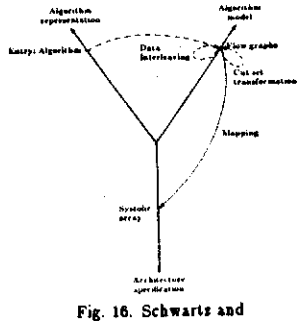

Fig. 16. Schwartz and Barnwell's method

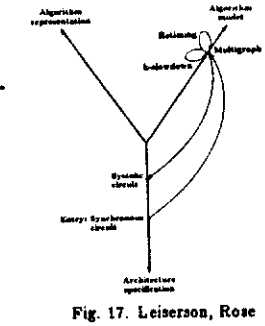

Fig. 17. Leiserson, Rose and Saxe's method

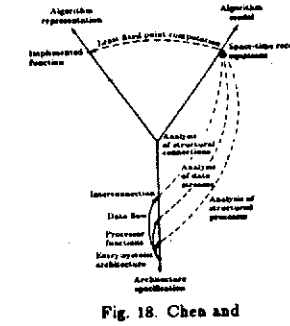

Fig. 18. Chen and Mead's method