General-Purpose Systolic Arrays

Kurtis T. Johnson and A.R. Hurson, Pennsylvania State University
Behrooz Shirazi, University of Texas, Arlington

Systolic arrays effectively exploit massive parallelism in computationally intensive applications. With advances in VLSI, WSI, and FPGA technologies, they have progressed from fixed-function to general-purpose architectures.

When Sun Microsystems introduced its first workstation, the company could not have imagined how quickly workstations would revolutionize computing. The idea of a community of engineers, scientists, or researchers time-sharing on a single mainframe computer could hardly have become ancient any more quickly. The almost instant wide acceptance of workstations and desktop computers indicates that they were quickly recognized as giving the best and most flexible performance for the dollar.

Desktop computers proliferated for three reasons. First, very large scale integration (VLSI) and wafer scale integration (WSI), despite some problems, increased the gate density of chips while dramatically lowering their production cost.[1] Moreover, increased gate density permits a more complicated processor, which in turn promotes parallelism.

Second, desktop computers distribute processing power to the user in an easily customized open architecture. Real-time applications that require intensive I/O and computation need not consume all the resources of a supercomputer. Also, desktop computers support high-definition screens with color and motion far exceeding those available with any multiple-user, shared-resource mainframe.

Third, economical, high-bandwidth networks allow desktop computers to share data, thus retaining the most appealing aspect of centralized computing, resource sharing. Moreover, networks allow the computers to share data with dissimilar computing machines. That is perhaps the most important reason for the acceptance of desktop computers, since all the performance in the world is worth little if the machine is isolated.

Today's workstations have redefined the way the computing community distributes processing resources, and tomorrow's machines will continue this trend with higher bandwidth networks and higher computational performance. One way to obtain higher computational performance is to use special parallel coprocessors to perform functions such as motion and color support of high-definition screens. Future computationally intensive applications suited for desktop computing machines include real-time text, speech, and image processing. These applications require massive parallelism.[2]

0018-9162/93/1100-0020$03.00 © 1993 IEEE

Many computational tasks are by their very nature sequential; for other tasks the degree of parallelism varies. Therefore, a massively parallel computational architecture must maintain sufficient application flexibility and computational efficiency. It must be[3]

- reconfigurable to exploit application-dependent parallelisms,
- high-level-language programmable for task control and flexibility,
- scalable for easy extension to many applications, and
- capable of supporting single-instruction stream, multiple-data stream (SIMD) organizations for vector operations and multiple-instruction stream, multiple-data stream (MIMD) organizations to exploit nonhomogeneous parallelism requirements.

Systolic arrays are ideally qualified for computationally intensive applications. Whether functioning as a dedicated fixed-function graphics processor or a more complicated and flexible coprocessor shared across a network, a systolic array effectively exploits massive parallelism. Falling into an area between vector computers and massively parallel computers, systolic arrays typically combine intensive local communication and computation with decentralized parallelism in a compact package. They capitalize on regular, modular, rhythmic, synchronous, concurrent processes that require intensive, repetitive computation. While systolic arrays originally were used for fixed or special-purpose architectures, the systolic concept has been extended to general-purpose SIMD and MIMD architectures.

Why systolic arrays?

Ever since Kung proposed the systolic model,[4] its elegant solutions to demanding problems and its potential performance have attracted great attention. In physiology, the term *systolic* describes the contraction (systole) of the heart, which regularly sends blood to all cells of the body through the arteries, veins, and capillaries. Analogously, systolic computer processes perform operations in a rhythmic, incremental, cellular, and repetitive manner. The systolic computational rate is restricted by the array's I/O operations, much as the heart controls blood flow to the cells since it is the source and destination for all blood.[4]

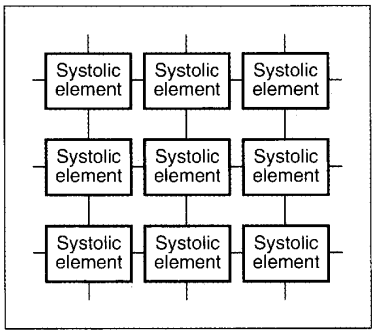


Figure 1. General systolic organization.

Although there is no widely accepted standard definition of systolic arrays and systolic cells, the following description serves as a working definition in this article (also see "Systolic array summary" below). Systolic arrays have balanced, uniform, gridlike architectures (Figure 1) in which each line indicates a communication path and each intersection represents a cell or a systolic element. However, systolic arrays are more than processor arrays that execute systolic algorithms. Systolic arrays consist of elements that take one of the following forms:

- a special-purpose cell with hardwired functions,
- a vector-computerlike cell with an instruction decoding unit and a processing unit, or
- a processor complete with a control unit and a processing unit.

In all cases, the systolic elements or cells are customized for intensive local communications and decentralized parallelism. Because an array consists of cells of only one or, at most, a few kinds, it has regular and simple characteristics. The array usually is extensible with minimal difficulty.

Three factors have contributed to the systolic array's evolution into a leading approach for handling computationally intensive applications: technology advances, concurrency processing, and demanding scientific applications.[4]

Technology advances. Advances in

Systolic array summary

Systolic array: A gridlike structure of special processing elements that processes data much like an *n*-dimensional pipeline. Unlike a pipeline, however, the input data as well as partial results flow through the array. In addition, data can flow in a systolic organization at multiple speeds in multiple directions. Systolic arrays usually have a very high rate of I/O and are well suited for intensive parallel operations.

Applications: Matrix arithmetic, signal processing, image processing, language recognition, relational database operations, data structure manipulation, and character string manipulation.

Special-purpose systolic array: An array of hardwired systolic processing elements tailored for a specific application. Typically, many tens or hundreds of cells fit on a single chip.

General-purpose systolic array: An array of systolic processing elements that can be adapted to a variety of applications via programming or reconfiguration.

Programmable systolic array: An array of programmable systolic elements that operates either in SIMD or MIMD fashion. Either the arrays interconnect or each processing unit is programmable and a program controls dataflow through the elements.

Reconfigurable systolic array: An array of systolic elements that can be programmed at the lowest level. FPGA (field-programmable gate array) technology allows the array to emulate hardwired systolic elements at a very low level for each unique application.

VLSI/WSI technology complement the systolic array's qualifications in the following ways:

- Smaller and faster gates allow a higher rate of on-chip communication because data has a shorter distance to travel.
- Higher gate densities permit more complicated cells with higher individual and group performance. Granularity increases as word length increases, and concurrency increases with more complicated cells.
- Economical design and fabrication processes produce less expensive systolic chips, even in small quantities. Better design tools allow arrays to be designed more efficiently. A systolic cell can be fully simulated before fabrication, reducing the chances that it will fail to work as designed. With advances in simulation techniques, fully tested, unique cells can now be quickly copied and arranged in regular, modular arrays. As VLSI/WSI designs become more complicated, "systolicizing" them provides an efficient way to ensure fault tolerance; any fault tolerance precautions built into one cell are extensible to all cells.
- Relatively new field-programmable gate array (FPGA) technology permits a reconfigurable architecture, as opposed to a reprogrammable architecture.

Concurrency processing. Past efforts to add concurrency to the conventional, von Neumann computer architecture have yielded coprocessors, multiple processing units, data pipelining, and multiple homogeneous processors. Systolic arrays combine features from all of these architectures in a massively parallel architecture that can be integrated into existing platforms without a complete redesign. A systolic array can act as a coprocessor, can contain multiple processing units and/or processors, and can act as an n-dimensional pipeline. Although data pipelining reduces I/O requirements by allowing adjacent cells to reuse the input data, the systolic array's real novelty is its incremental instruction processing or computational pipelining.[5] Each cell computes an incremental result, and the computer derives the complete result by interpreting the incremental results from the entire array in a prespecified algorithmic format.

Demanding scientific applications. The technology growth of the last three decades has produced computing environments that make it feasible to attack demanding scientific applications on a larger scale. Large-matrix multiplication, feature extraction, cluster analysis, and radar signal processing are only a few examples.[5] As recent history shows, when many computer users work on a wide variety of applications, they develop new applications requiring increased computational performance. Examples of these innovative applications include interactive language recognition, relational database operations, text recognition, and virtual reality.[5] These applications require massive repetitive and rhythmic parallel processing, as well as intensive I/O operation. Hence, systolic computing.

Implementation issues

A number of implementation issues determine a systolic array's performance efficiency. Designers should understand the following performance trade-offs at the design stage.

Algorithms and mapping. Designers must be intimately familiar with the algorithms they are implementing on systolic arrays. Designing a systolic array heuristically from an algorithm is slow and error-prone, requiring simulation for verification and often producing a less-than-optimum algorithm. Thus, automatic array synthesis is an important research area.[6] At present, however, most array designs are based on heuristics.

Integration into existing systems. Generally, a systolic array is integrated into an existing host as a back-end processor. The array's high I/O requirements often make system integration a significant problem. Because the existing I/O channel rarely satisfies the array's bandwidth requirement, a memory subsystem often must be added between the host and the systolic array to support data access and data multiplexing and demultiplexing. The memory subsystem can range from the complicated support and cluster processors in the Warp array to the simpler staging memory in the Splash array. (These systems will be discussed in greater detail later.)

Cell granularity. The level of cell granularity directly affects the array's throughput and flexibility and determines the set of algorithms that it can efficiently execute. Each cell's basic operation can range from a logical or bitwise operation, to a word-level multiplication or addition, to a complete program. Granularity is subject to technology capabilities and limitations as well as design goals. For example, integration-substrate families have different performance and density characteristics. Packaging also introduces I/O pin restrictions.

Extensibility. Because systolic arrays are built of cellular building blocks, the cell design should be sufficiently flexible for use in a wide variety of topologies implemented in a wide variety of substrate technologies.

Clock synchronization. Clock lines of different lengths within integrated chips, as well as external to the chips, can introduce skews. The risk of clock skew is greater when dataflow in the systolic array is bidirectional. Wavefront arrays[5] reduce the clock skew problem by introducing more complicated, asynchronous, intercellular communications.

Reliability. As integrated circuits grow larger, designers must build in greater fault tolerance to maintain reliability, and diagnostics to verify proper operation.

Systolic array taxonomy

The term *systolic array* originally referred to special-purpose or fixed-function architectures designed as hardware implementations of a given algorithm. In mass quantities, the production of these arrays was manageable and economical, and, thus, they were well suited for common applications. But these designs were bound to the specific application at hand and were not flexible or versatile. Every time a systolic array was to be used on a new application, the manufacturer had to undertake the long, costly, and potentially risky process of designing, testing, and fabricating an application-specific integrated chip. Although the cost and risks of developing ASICs have decreased in recent

Table 1. Systolic array taxonomy.

Class	General-purpose										Special-purpose
Type	Programmable			Reconfigurable			Hybrid				Hardwired
Organization	SIMD or MIMD			VFIMD							VFIMD
Topology	Programmable		Fixed	Reconfigurable		Fixed	Hybrid			Fixed	Fixed
Interconnections	Static	Dynamic	Fixed	Static	Dynamic	Fixed	Static	Dynamic		Fixed	Fixed
Dimensions	n-dimensional (n > 2 is rare due to complexity)										n-dimensional

SIMD: single-instruction stream, multiple-data stream; MIMD: multiple-instruction stream, multiple-data stream; VFIMD: very-few-instruction stream, multiple-data stream.

years, budget constraints have motivated a trend away from unique hardware development. Consequently, general-purpose systolic architectures have become a logical alternative. In addition to serving in a wide variety of applications, they also provide test beds for developing, verifying, and debugging new systolic algorithms. Table 1 shows a taxonomy of general-purpose and special-purpose systolic arrays.

Special-purpose architectures. Special-purpose systolic architectures are custom designed for each application. Few problems resist attack from systolic arrays, but some problems may require elegant algorithms. Generally speaking, the systolic design requires a performance algorithm that can be efficiently implemented with today's VLSI technology.

One area that easily utilizes systolic algorithms is matrix operations. Figure 2 illustrates the algorithm for the sum of a scalar product, computed in a single systolic element. After the cell is initialized, the a's and the b's are synchronously shifted through the processing element. The accumulator stores the sum of the a,b products. All the a and b data synchronously exits the processing element unmodified to be available for the next element. At the end of processing, the sum of the products is shifted out of the accumulator. This principle easily extends to a matrix product, as shown in Figure 3. The only difference between single-element processing and array processing is that the latter delays each additional column and row by one cycle so that the columns and rows line up for a matrix multiply. The product matrix is shifted out after completion of processing.

An obvious problem with this approach is that matrix products involving a matrix larger than the systolic array must be divided into a set of smaller matrix products. This resource and implementation problem affects all systolic arrays. Proper algorithm development compensates for the problem, but performance decreases nevertheless.

The matrix product example also demonstrates another problem with special-purpose systolic arrays and hardware in

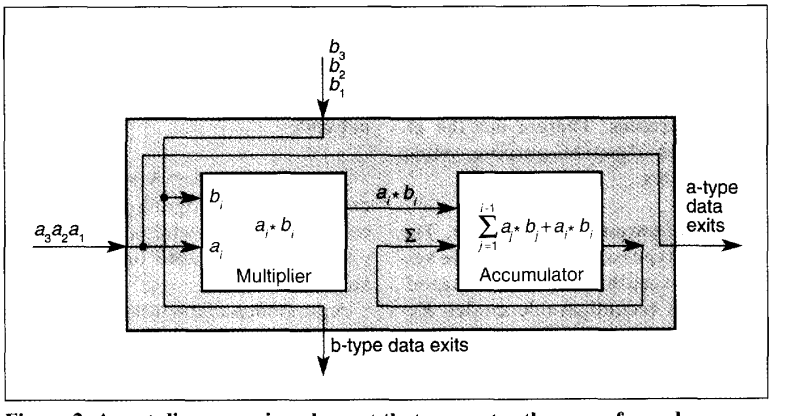


Figure 2. A systolic processing element that computes the sum of a scalar product.

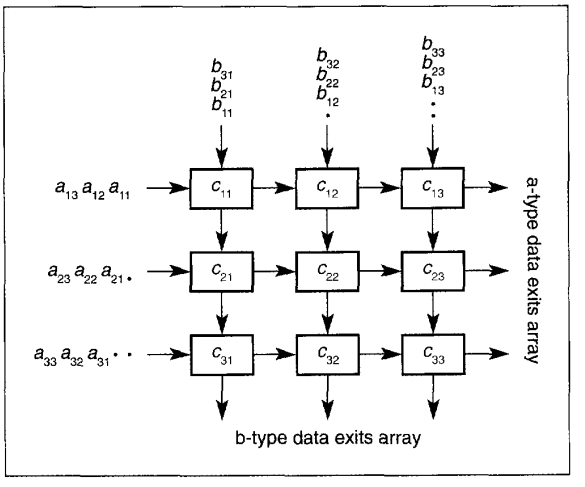


Figure 3. The systolic product of two 3 × 3 matrices.

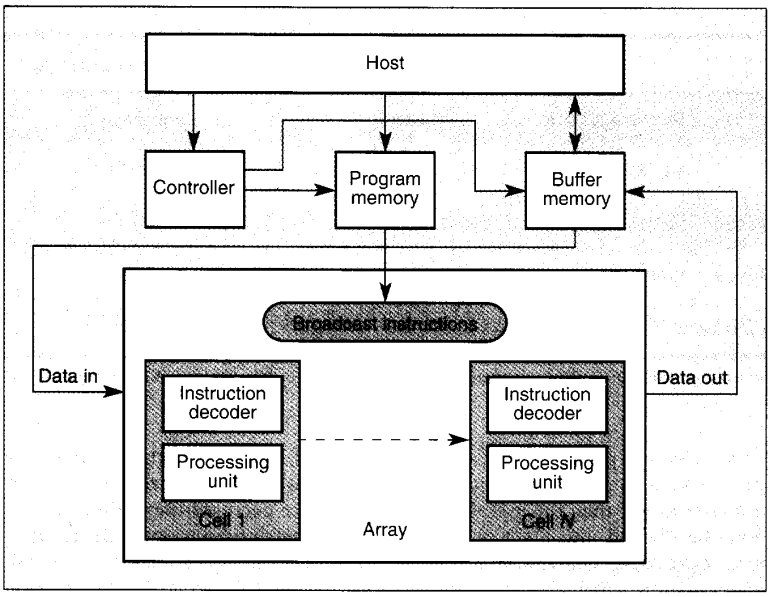


Figure 4. General organization of SIMD programmable linear systolic arrays.

general. The more specialized the hardware, the higher the performance; but cost per application also rises and flexibility decreases. Therein lies the attractiveness of general-purpose systolic architectures.

General-purpose architectures. The two basic types of general-purpose systolic arrays are the programmable model and the reconfigurable model. Recently, hybrid models have also been proposed.

In the programmable model, cell architectures and array architectures remain the same from application to application. However, a program controls data operations in the cells and data routing through the array. All communication paths and functional units are fixed, and the program determines when and which paths are utilized. One of the first programmable architectures was Carnegie Mellon's programmable systolic chip.

In the reconfigurable model, cell architectures as well as array architectures change from one application to another. The architecture for each application appears as a special-purpose array. The primary means of implementing the reconfigurable model is FPGA technology. Splash was one of the early FPGA-based reconfigurable systolic arrays.

Hybrid models make use of both VLSI and FPGA technology. They usually consist of VLSI circuits embedded in an FPGA-reconfigurable interconnection network.

Systolic topologies. Array topologies can be either programmable or reconfigurable. Likewise, array cells are either programmable or reconfigurable.

A programmable systolic architecture is a collection of interconnected, general-purpose systolic cells, each of which is either programmable or reconfigurable. Programmable systolic cells are flexible processing elements specially designed to meet the computational and I/O requirements of systolic arrays. Programmable systolic architectures can be classified according to their cell interconnection topologies: fixed or programmable.

Fixed cell interconnections limit a given topology to some subset of all possible algorithms. That topology can emulate other topologies by means of the proper mapping transformation, but reduced performance is often a consequence.

Programmable cell interconnection topologies typically consist of programmable cells embedded in a switch lattice that allows the array to assume many different topologies. Programmable topologies are either static or dynamic. Static topologies can be altered between applications, and dynamic topologies can be altered within an application. Static programmable topologies can be implemented with much less complexity than dynamic programmable topologies. There has been little research in dynamic programmable topologies because a highly complex interconnection network could undermine the regular and simple principles of systolic architectures.

Reconfigurable systolic architectures capitalize on FPGA technology, which allows the user to configure a low-level logic circuit for each cell. Reconfigurable arrays also have either fixed or reconfigurable cell interconnections. The user reconfigures an array's topology by means of a switch lattice. Any general-purpose array that is not conventionally programmable is usually considered reconfigurable. All FPGA reconfiguring is static due to technology limitations.

Array dimensions. We can further classify general-purpose and special-purpose systolic architectures by their array dimensions. The two most common structures are the linear array and the two-dimensional array. Linear systolic arrays are by default statically reconfigurable in one-dimensional space. Two-dimensional arrays allow more efficient execution of complicated algorithms. Due to I/O limitations, general-purpose systolic arrays of dimensions greater than two are not common.

Programmable array organization. Programmable systolic arrays are programmable either at a high level or a low level. At either level, programmable arrays can be categorized as either SIMD or MIMD machines. They are typically back-end processors with an additional buffer memory to handle the high systolic I/O rates. High-level programmable arrays usually are programmed in high-level languages and are word oriented. Low-level arrays are programmed in assembly language and are bit oriented.

SIMD. SIMD systolic machines (Figure 4) operate similarly to a vector processor. The host workstation preloads a controller and a memory, which are external to the array, with the instructions and data for the application. The systolic cells store no programs or instructions. Each cell's instruction-process-

ing functional unit serves as a large decoder. Once the workstation enables execution, the controller sequences through the external memory, thus delivering instructions and data to the systolic array. Within the array, instructions are broadcast and all cells perform the same operation on different data. Adjacent cells may share memory, but generally no memory is shared by the entire array. After exiting the array, data is collected in the external buffer memory. SIMD-programmable systolic cells take less space on the VLSI wafer than their MIMD counterparts because they are simple instruction-processing elements requiring no program memory. From tens to hundreds of such cells fit on one integrated chip.[7]

MIMD. MIMD systolic machines (Figure 5) operate similarly to homogeneous von Neumann multiprocessor machines. The workstation downloads a program to each MIMD systolic cell. Each cell may be loaded with a different program, or all the cells in the array may be loaded with the same program. Each cell's architecture is somewhat similar to the conventional von Neumann architecture: It contains a control unit, an ALU, and local memory. MIMD systolic cells have more local memory than their SIMD counterparts to support the von Neumann-style organization. Some may have a small amount of global memory, but generally no memory is shared by all the cells. Whenever data is to be shared by processors, it must be passed to the next cell. Thus, data availability becomes a very important issue. High-level MIMD systolic cells are very complicated, and usually only one fits on a single integrated chip. For example, the iWarp is a Warp cell without memory on a single chip. Local memory for each iWarp cell must be supplied by additional chips.

Reconfigurable array organization. Recent gate-density advances in FPGA technology have produced a low-level, reconfigurable systolic array architecture that bridges the gap between special-purpose arrays and the more versatile, programmable, general-purpose arrays. The FPGA architecture is unusual because a single hardware platform can be logically reconfigured as an exact duplicate of a special-purpose systolic array. Figure 6 shows the general organization of a reconfigurable array.

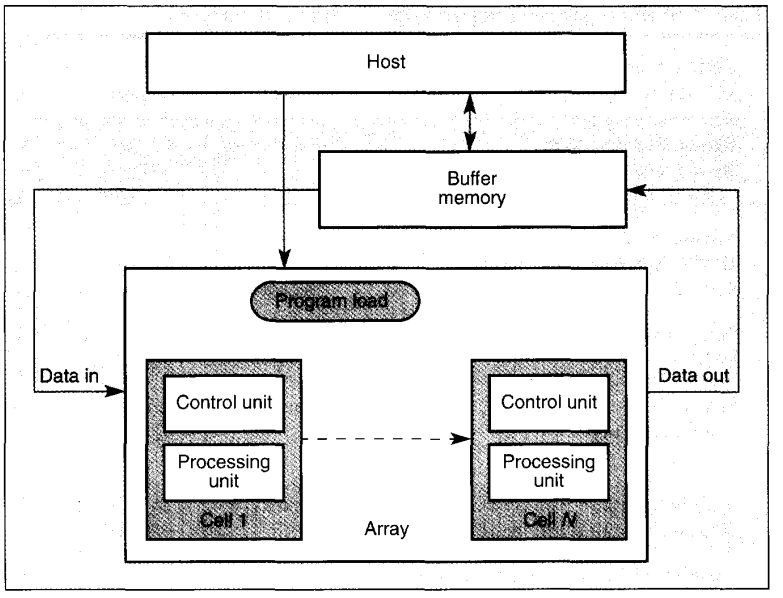


Figure 5. General organization of MIMD programmable linear systolic arrays.

The designer logically draws cell architectures on the workstation with a schematic editor (such as Mentor), converts the design to FPGA code with another utility, and then downloads the code in a few seconds to configure the FPGA architecture.[8]

Reconfigurable systolic arrays do not fall into the SIMD or MIMD categories for the same reason that special-purpose systolic arrays do not. Reconfigurable arrays, like special-purpose arrays, are generally limited to VFIMD (very-few-instruction streams, multiple-data streams) organization due to FPGA gate density. Instructions are implicit in the configuration of each cell; therefore, there is no need to download them from the workstation. Since special-purpose systolic architectures consist of a very few unique cells repeated throughout the array, the entire array also tends to be VFIMD. Purely reconfigurable architectures are fine-grain, low-level devices best suited for logical or bit manipulations. They typically lack the gate density to support high-level functions such as multiplication.

Tables 2, 3, and 4 list most of the recent programmable and reconfigurable, general-purpose systolic arrays reported in the literature. When information is unclear in the literature, the corresponding space in the table is left blank. The tables indicate three stages of product development:

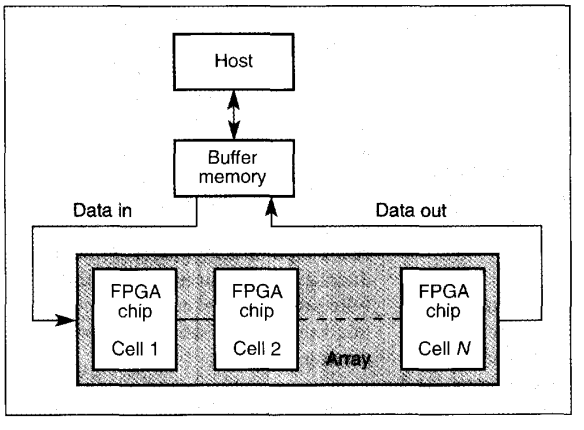


Figure 6. General organization of reconfigurable arrays.

Table 2. SIMD-organized programmable systolic architectures.

System name, developer	Development stage	Topology	Key features
Brown Systolic Array[1] Brown University	Prototype	Linear 470 cells	Very small VLSI footprint; 100s of cells per chip; ISA, SSR architectures; 8-bit ALU only; 157 MOPS
Micmacs[2] IRISA, Campus de Beaulier, France	Prototype	Linear 18 cells	16-bit fixed-point math; broadcast data; 90 MOPS
Geometric Arithmetic Parallel Processor[3] NCR	Commercial	48 × 48 array 2,304 cells	Bit-slice cellular architecture; global data; 900 MOPS
Saxpy-1M[4] Computer Corp.	Commercial	Linear 32 cells	32-bit floating-point capability; broadcast and Saxpy global data with block processing 1,000 MFLOPS
Systolic/Cellular Architecture[5] Hughes Research Laboratories	Prototype	16 × 16 array 256 cells	32-bit fixed-function units
Cylindrical Banyan Multicomputer[6] University of Texas at Austin	Research		Packet-switched programmable topology with programmable cells
Programmable Systolic Device[7] Australian National University	Research		Instruction decoding occurs once per chip; each chip has many cells

1. R. Hughey and D. Lopresti, "Architecture of a Programmable Systolic Array," *Proc. Int'l Conf. Systolic Arrays*, IEEE CS Press, Los Alamitos, Calif., Order No. 860, 1988, pp. 41-49.
2. P. Frison et al., "Micmacs: A VLSI Programmable Systolic Architecture," *Systolic Array Processors*, Prentice-Hall, Englewood Cliffs, N.J., 1989, pp. 145-155.
3. P. Greussay, "Programmation des Mega-Processeurs: du GAPP à la Connection Machine Course Notes DEA Artificial Intelligence," Paris Univ., VIII, Vincennes, 1985.
4. D. Foulser and R. Scheiber, "The Saxpy Matrix-1: A General-Purpose Systolic Computer," *Computer*, Vol. 20, No. 7, July 1987, pp. 35-43.
5. K.W. Przytula and J.B. Nash, "Implementation of Synthetic Aperture Radar Algorithms on a Systolic/Cellular Architecture," *Proc. Int'l Conf. Systolic Arrays*, IEEE CS Press, Los Alamitos, Calif., Order No. 860, 1988, pp. 21-27.
6. M. Malek and E. Opper, "The Cylindrical Banyan Multicomputer: A Reconfigurable Systolic Architecture," *Parallel Computing*, Vol. 10, No. 3, May 1989, pp. 319-326.
7. P. Lenders and H. Schroder, "A Programmable Systolic Device for Image Processing Based on Morphology," *Parallel Computing*, Vol. 13, No. 3, Mar. 1990, pp. 337-344.

- Research: A functional system that has not yet been implemented.
- Prototype: At least a partial system has been implemented.
- Commercial: The system has been implemented, and an integrated chip with multiple cells, a complete system, or both are available for purchase.

Architectural issues of programmable systolic arrays

To be effective, a programmable systolic architecture must adhere to the general principles of systolic arrays: regularity, simplicity, concurrency, and rhythmic communications. In addition, the introduction of a programmable systolic cell with significant local memory has resulted in a new mode of operations: block processing. Block processing combines periods of intensive systolic I/O and periods of sequential von Neumann-style processing to form what is known as a pseudosystolic model.

Replicating and interconnecting a basic programmable systolic cell to form an array carries out the regularity and simplicity principles, provided an appropriate algorithm and program are utilized. However, programmable-cell-based architectures must introduce special features to address the systolic properties of concurrency, rhythmic communications, and block processing.

Concurrency. Two levels of concurrency are possible in a programmable systolic architecture: concurrency across cells and concurrency within cells, both implicit to hardwired designs. To make concurrency feasible in programmable systolic architectures, the designer must add special mechanisms that facilitate processing control.

Intercell concurrency. Coordination of concurrency control has always been inherent in a hardwired systolic design because it has been addressed at the algorithm development stage. The advent of the programmable systolic array required that innovative features be incorporated into designs to satisfy a programmable coordination requirement. Methods of program loading, memory initialization, program switching, and local memory access at the cell level had

Table 3. MIMD-organized programmable systolic architectures.

System name, developer	Development stage	Topology	Key features
Programmable Systolic Chip Architecture[1] Carnegie Mellon University	Prototype	Linear 9 cells	Early predecessor of Warp; 8-bit fixed-point ALUs
Warp[2] Carnegie Mellon University	Commercial	Linear 10 cells	32-bit floating-point multiplication; block processing; I/O queuing, 100 MFLOPS
iWarp[3] Carnegie Mellon University	Prototype	8 × 8 array	Warp cell minus on-chip memory; expandable to 1,024 cells
Computer for Experimental Synthetic Aperture Radar[4] Norwegian Defense Res. Estab.	Prototype	Four 8 × 16 arrays 512 cells	Bit-serial cellular I/O; 32-bit floating-point multipliers in each cell; 320 MFLOPS
Cellular Array Processor[5] Fujitsu Laboratories, Japan	Commercial	16 × 16 array 256 cells	Block processing; floating-point math; image oriented
Configurable Highly Parallel Computer[6] Purdue University	Research		Programmable cells embedded in switch lattice for programmable topology
Associative String Processor[7] Brunel University, UK	Research		Associative concepts; MIMD at high level; SIMD at low level
PICAP3[8] University of Paris	Prototype	8 × 8 array 64 cells	16-bit word length; image oriented
PSDP[9] Univ. of South Florida, Honeywell	Research		4-bit word length; wafer-scale design

1. A.L. Fisher, K. Sarocky, and H.T. Kung, "Experience with the CMU Programmable Systolic Chip," *Proc. Soc. of Photo-Optical Instrumentation Engineers, Real-Time Signal Processing VII*, Spie, San Diego, Calif., 1984, p. 495.
2. M. Annaratone et al., "Architecture of Warp," *Proc. Compcon*, IEEE CS Press, Los Alamitos, Calif., Order No. 764, Feb. 1987, pp. 264-267.
3. S. Borkar et al., "iWarp: An Integrated Solution to High-Speed Parallel Computing," *Proc. Supercomputing*, IEEE CS Press, Los Alamitos, Calif., Order No. 882, 1988, pp. 330-339.
4. M. Toverud and V. Anderson, "CESAR: A Programmable High-Performance Systolic Array Processor," *Proc. Int'l Conf. Computer Design*, IEEE CS Press, Los Alamitos, Calif., Order No. 872 (microfiche only), 1988, pp. 414-417.
5. M. Ishii et al., "Cellular Array Processor CAP and Application," *Proc. Int'l Conf. Systolic Arrays*, IEEE CS Press, Los Alamitos, Calif., Order No. 860, 1988, pp. 535-544.
6. L. Snyder, "Introduction to the Configurable Highly Parallel Computer," *Computer*, Vol. 15, No. 1, Jan. 1982, pp. 47-56.
7. R.M. Lea, "The ASP, a Fault-Tolerant VLSI/ULSI/WSI Associative String Processor for Cost-Effective Processing," *Proc. Int'l Conf. Systolic Arrays*, pp. 515-524.
8. B. Lindscog and P.E. Danielsson, "PICAP3: A Parallel Processor Tuned for 3D Image Operations," *Proc. Eighth Int'l Conf. Pattern Recognition*, IEEE CS Press, Order No. 742 (microfiche only), 1986, pp. 1248-1250.
9. D. Landis et al., "A Wafer-Scale Programmable Systolic Data Processor," *Proc. Ninth Biennial Univ./Gov't/Ind. Microelectronics Symp.*, IEEE, Piscataway, N.I., 1991, pp. 252-256.

to be incorporated into a systolic environment. The following mechanisms facilitate programmable concurrency throughout the array:

- *Broadcast data* permits all cells of an array to be reset simultaneously with initial conditions and constants at the start of a processing block during nonsystolic modes of operation. The larger the array, the more efficiency it will gain from a broadcast data capability.
- *Broadcast instructions* permit operations similar to those of a vector computer by making each systolic cell analogous to the vector computer's processing unit. However, unlike most vector computers, systolic cells can support a high-bandwidth communication channel with adjacent cells.
- *Broadcast instruction addresses* allow fast and efficient switching among programs in an MIMD cell's program memory. When all MIMD cells have programs stored in the same places in their program memories, a jump to the broadcast instruction address carries out simultaneous program switching throughout the array. If the programs in all the MIMD cells happen

Table 4. VFIMD-organized programmable systolic architectures.

System name, developer	Development stage	Topology	Key features
Splash Systolic Engine[1] Super Computing Research Center	Prototype	Linear 32 cells	FPGA-reconfigurable cell architecture based on commercial chip
Cellular Array Logic[2] Edinburgh University, UK	Prototype	16 × 16 array 256 cells	FPGA-reconfigurable cell architecture based on custom chip
Hybrid Architecture[3] University of Illinois	Research		Reconfigurable cell architecture integrating FPGA capability and 32-bit floating-point multiplication
Programmable Adaptive Computing Engine[4] University of Wales, Bangor, UK	Prototype	Programmable	Functional units embedded in each cell; reconfigurable connections in cell as well as programmable topology
Configurable Functional Array[5] Tsinghua University, Beijing	Research		Configurable array topology and cell architecture

1. M. Gokhale et al., "Splash: A Reconfigurable Linear Logic Array," *Proc. Int'l Conf. Parallel Processing*, IEEE CS Press, Los Alamitos, Calif., Order No. 2101, 1990, pp. 1526-1531.
2. T. Kean and J. Gray, "Configurable Hardware: Two Case Studies of Micrograin Computation," in *Systolic Array Processors*, Prentice-Hall, Englewood Cliffs, N.J., 1989, pp. 310-319.
3. R. Smith and G. Sobelman, "Simulation-Based Design of Programmable Systolic Arrays," *Computer-Aided Design*, Vol. 2.3, No. 10, Dec. 1991, pp. 669-675.
4. S. Jones, A. Spray, and A. Ling, "A Flexible Building Block for the Construction of Processor Arrays," in *Systolic Array Processors*, Prentice-Hall, Englewood Cliffs, N.J., 1989, pp. 459-466.
5. C. Wenyang, L. Yanda, and J. Yue, "Systolic Realization for 2D Convolution Using Configurable Functional Method in VLSI Parallel Array Designs," *Proc. IEEE, Computers and Digital Technology*, Vol. 138, No. 5, Sept. 1991, pp. 361-370.

to be identical, the array is performing SIMD operations.

- *Instruction systolic arrays*[9] provide a precise low-level method of coordinating processing from cell to cell. In an ISA, instructions travel through the array with the data. The processed data passes with the original instruction from each cell to the next. An appropriately wide ISA instruction also has built-in microcodable parallelism. ISA algorithms can be lengthy and more difficult to develop than algorithms for other SIMD and MIMD arrays. But they provide a way to uniquely specify concurrency across cells without an overly complicated circuit.
- *Direct local memory access or global memory*. Status flags can be stored either in each cell's local memory or in a global memory. In the case of local memory, the controller must be able to directly access each cell's local memory to determine flag status; otherwise, the controller must issue a request and wait for it to propagate through the array. When global memory is available, the controller obtains status information much more easily. Direct local memory access or global memory also facilitates initialization within a cell. The Saxpy-1M is one of the first systolic arrays with global memory.

Intracell concurrency. Programmable cells require the ability to perform multiple similar or dissimilar operations simultaneously. Mechanisms incorporated into programmable systolic cells to support concurrency are all proven techniques that originated in conventional von Neumann processors: microcodable processing elements, duplication of functional units, multiple data paths, sufficiently wide instruction words, and pipelining of functional units.

Rhythmic communications. The principle of rhythmic communications clearly separates systolic arrays from other architectures. Programmability creates a more flexible systolic architecture, but the penalties are complexity and possible slowing of operations. When systolic cells are programmable, the issue of data availability arises. To minimize the performance degradation of programmable systolic designs, each cell's processing element must have data available when it is required, high-speed arithmetic capabilities, and the ability to transfer or store the processed data. Thus, each cell must have sufficient memory for data storage as well as program storage to facilitate intercell communications.

The following architectural features support rhythmic communications:

- *Queued I/O* streamlines cellular communications by allowing the source cell to send data to the destination cell when the data is available, not when the destination cell is ready to accept it. Data exits the queue on a FIFO (first in, first out) basis that allows program computation to proceed irrespective of communications status. Queued I/O helps considerably when all cells are computing a single program that is skewed in time across the cells. A disadvantage is that this mechanism uses memory space that is expensive on VLSI wafers.
- *Serial I/O* reduces the bandwidth requirement on the systolic array's workstation machine, at the same time promoting rhythmic cellular communication. The resulting pro-

grammable systolic array has bit-serial cellular communications and word-wide operations in each cell. Each cell stores partial words until it receives the full word and then performs functional operations. Using more cells can partially offset the disadvantage of reduced throughput.

- *Multiple data paths* provide the cell with fast, parallel internal and external communications. One or more data paths are dedicated to computational needs, another to cellular input operations, and yet another to cellular output operations.
- *Systolic shared registers* provide several benefits specifically for programmable systolic architectures. SSR architectures (Figure 7) provide a program-implicit means of streamlined dataflow. Each two adjacent processing elements share a small register memory. Data flows concurrently with computation as each processing element receives new data from the register memory upstream, operates on it, and directs it to the next register memory downstream. This architecture eliminates the requirement that data movement be explicitly specified, and dataflow through the array becomes a result of processing. The input register and the output register are specified by the instruction word. Therefore, bidirectional dataflow is a natural result. An SSR architecture is advantageous over queued I/O because communications do not occur on a FIFO basis. A disadvantage is that the register bank must be kept small to minimize access time and thus maximize systolic bandwidth. The small register makes programming difficult for some of the more complicated algorithms.
- *Broadcast data* eases systolic I/O requirements in certain instances by allowing all functional units and local memory of a cell to be initialized or set to a common variable at once.
- *Global data* eases systolic I/O requirements and the cell's storage requirements by keeping only one copy of the same data and allowing all cells to share it. With proper global memory coherency, any modifications to global data are instantly available to other cells.

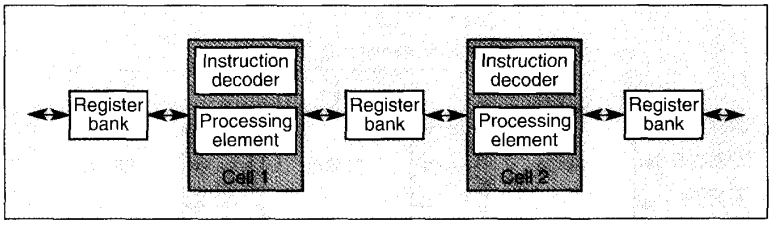


Figure 7. An SIMD systolic shared-register architecture.

- *Bidirectional and wraparound dataflow* must be explicitly specified in the program, whereas in hardwired designs dataflow is implicit. Bidirectional and wraparound dataflow can reduce the I/O bandwidth requirements of the external buffer memory by recirculating data within the array. More flexible dataflow also allows arrays to execute a wide variety of algorithms more efficiently.

Block processing. If each programmable systolic cell has significant local memory, block processing is possible in the systolic environment. During block processing, very little systolic I/O occurs, and each cell executes a series of instructions on local data. Once the cells generate an intermediate result, processing pauses while systolic I/O takes place, and new data is written to each cell's local memory.

Block processing on programmable systolic architectures can result in a more efficient, pseudosystolic operation in some applications for two reasons. First, the merger of von Neumann programmable cell architectures into a systolic array environment causes many of the same problems found in homogeneous multiprocessor systems. The serial nature of von Neumann machines interferes with the rhythmic systolic I/O of the array, causing an unacceptable amount of time wasted in waiting for data to become available. Block processing minimizes this wasted time in applications that can be divided into equal segments. Applications are divided into parallel tasks that utilize local data.

Second, for any systolic array, the bandwidth and size of the external memory are always the limiting factors on throughput and performance. Block processing reduces the array's systolic I/O requirements by reducing the amount of cellular I/O.

Important work has been done on characterizing block processing in a general-purpose systolic array.[10] Block processing can be performed on either SIMD or MIMD programmable systolic machines. Requirements for efficient block processing include an appropriate algorithm and significant local memory in each cell. Other types of array communication such as broadcast data and global data are also desirable.

Architectural issues of reconfigurable systolic arrays

The primary architectural issues in designing a reconfigurable systolic array are the hardware platform of FPGAs and the class of algorithms to be targeted to that platform. The number of FPGAs, their topology, and their gate density will determine the set of systolic architectures that can be synthesized and the set of systolic algorithms that can be implemented on that hardware platform. Reconfigurable systolic architectures are very interesting because the cell's architecture is actually programmed. Consequently, the architecture programmer has complete control over what architectural features are incorporated in each FPGA. Determining the cell architecture can be an iterative process that continuously refines the architecture until it satisfies application requirements.

FPGA architecture programming differs from conventional programming in that one programs the circuit's logical function, instead of programming a model of operations in a high-level language. Reconfiguring an FPGA is the same as logically drawing a new circuit. Therefore, an FPGA platform can assume any architecture that FPGA gate density and package pinout permit.

Communication and concurrency

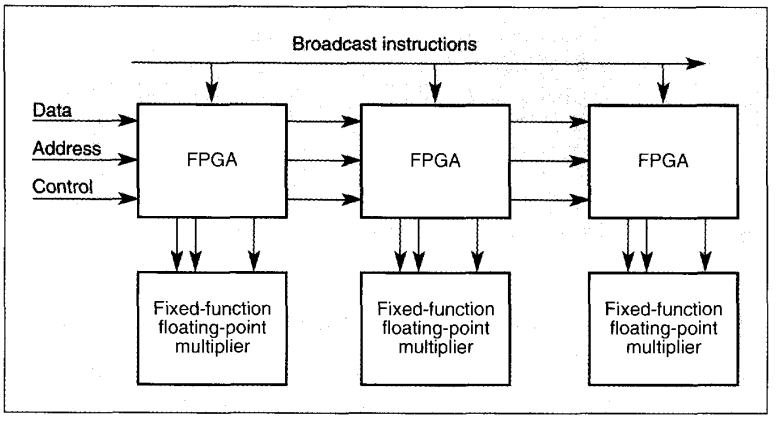


Figure 8. A hybrid SIMD programmable systolic architecture.

must be evaluated on a case-by-case basis at the time of configuration. Current gate-density technology does not make MIMD arrays feasible due to cell memory and control requirements, but FPGA platforms are well suited for logical or bit-level applications. Splash, for example, is a linear reconfigurable array appropriate for bit-level applications.

An FPGA chip can be configured for one medium-size systolic element or for several simple systolic cells. In either case, current technology limits the circuit size to an order of magnitude approaching 25,000 gates. Packaging and I/O pins also influence the design. Current technology limits I/O to approximately 200 pins.

Reconfigurable architectures are also highly qualified for fault-tolerant computing. Reconfiguration can simply eliminate a bad cell from the array. Cell architectures can be configured around VLSI defects.

Architectural issues of hybrid arrays

High-level programmable arrays require extensive efforts to map algorithms to high-level languages after algorithm development. When mapping is completed, the resulting system often is not as efficient as a system implemented in a fixed-function ASIC. On the other hand, low FPGA gate density makes it unlikely that large-grain tasks will ever be completely implemented in FPGAs or that reconfigurable arrays will replace conventional SIMD and MIMD architectures in the near future. Consequently, a natural step is to merge the two approaches, keeping their desirable aspects and discarding their undesirable aspects.

The hybrid SIMD architecture[11] (Figure 8) is best utilized for intensive floating-point computational applications but does not degrade in performance as much as high-level programmable arrays when significant logical or control operations are included. The hybrid design combines a commercial floating-point multiplier chip and an FPGA controller to form a systolic cell. The commercial multiplier is used for its economy, speed, and package density; the FPGA closely binds the cell to a specific application. Another project is attempting to integrate a hybrid general-purpose systolic array cell on a single chip.[12]

A recently developed simulator allows simulation and testing of the cell and the array design for a hybrid architecture.[11] Unlike most commercially available computer-aided design utilities, which verify designs at the chip level, it verifies the performance of algorithms and architectures through simulation of the complete array. The simulator maintains the iterative nature of purely reconfigurable array design by facilitating tailoring of hybrid designs for specific applications.

Existing architectures

A review of projects initiated during the last decade shows that the trend was to develop large systolic array processors that require elaborate, customized host support. Warp, the Computer for Experimental Synthetic Aperture Radar, the Cellular Array Processor, and Saxpy-1M all require expensive and complicated I/O support — for the intensive instruction I/O as well as the data I/O. These machines are high-performance supercomputers (200 MFLOPS, 320 MFLOPS, and 1,000 MFLOPS, respectively) with centralized concurrency, constructed to fill an existing performance gap.

The introduction of workstations into the workplace has changed the way a significant portion of the computing community views computers. Users demand improved desktop computing performance as applications continue to increase in size and complexity. Workstation architecture can always be improved, especially for emerging applications such as text and speech processing and gene matching. The more recent general-purpose systolic array projects (Brown, Micmacs, Splash) show that back-end systolic processors are effective in boosting a workstation's performance for these applications. These arrays are small enough that the host's open architecture with limited I/O bandwidth does not severely impact the array's performance for low- to moderate-level granularity. Small back-end systolic processors are also economically sound. For example, Splash consists of a two-board add-on set for a Sun workstation. One board supports the linear array, and the other supports a buffer memory. The set costs from $13,000 to $35,000.[8]

Almost without exception, current research emphasizes reconfigurable cells, reconfigurable arrays, and hybrids of functional units embedded in reconfigurable FPGA arrays. Reconfigurable designs have proven to be unmatched for low- to moderate-granularity requirements but are not yet mature enough for high-granularity applications. In addition, as we have said, reconfigurable topologies and cells are highly fault-tolerant. Their fault tolerance is a configuration issue, not a design and fabrication issue.

Until FPGA chip density progresses to the point where a very large FPGA can achieve high-level granularity, hybrid architectures present perhaps the most practical means to a reconfigurable high-level systolic array. Hybrids make use of the most attractive features of programmable and reconfigurable

methods while adding greater flexibility than either method.

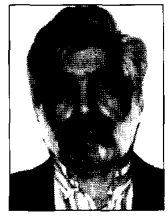
No discussion of general-purpose systolic arrays is complete without addressing the issues of programming and configuration. High-level-language programming is desirable for promoting widespread use of programmable systolic back-end processors. Of the projects we surveyed, only the larger systems had mature programming environments. Currently, the smaller programmable arrays have implemented only assembly programming. As mentioned earlier, FPGA-configurable arrays are configured with the use of a schematic editor. One drawback of this approach is that it typically takes more effort than programming a programmable cell.[7]

The systolic array is a formidable approach to exploiting concurrencies in a computationally rhythmic and intensive environment. General-purpose systolic arrays provide an economical way to enhance computational performance by emphasizing concurrency and parallelism. Arrays ranging from low-level to high-level granularity have been applied to problems from bit-oriented pixel mapping to 32-bit floating-point-based scientific computing. Systolic arrays hold great promise to be a pervasive form of concurrency processing. As a solution to the intensive computational performance requirements of tomorrow's applications, general-purpose systolic arrays cannot be overlooked. ■

References

1. W.K. Fuchs and E.E. Swartzlander Jr., "Wafer-Scale Integration: Architectures and Algorithms," *Computer*, Vol. 25, No. 4, Apr. 1992, pp. 6-8.
2. P. Quinton and Y. Robert, *Systolic Algorithms & Architectures*, Prentice-Hall, Englewood Cliffs, N.J., 1991.
3. A. Krikelis and R.M. Lea, "Architectural Constructs for Cost-Effective Parallel Computers," in *Systolic Array Processors*, Prentice-Hall, Englewood Cliffs, N.J., 1989, pp. 287-300.
4. H.T. Kung, "Why Systolic Architectures?" *Computer*, Vol. 15, No. 1, Jan. 1982, pp. 37-46.
5. J.A.B. Fortes and B.W. Wah, "Systolic Arrays: From Concept to Implementation," *Computer* (special issue on systolic arrays), Vol. 20, No. 7, July 1987, pp. 12-17.
6. C.K. Ko and O. Wing, "Mapping Strategy for Automated Design of Systolic Arrays," *Proc. Int'l Conf. Systolic Arrays*, IEEE CS Press, Los Alamitos, Calif., Order No. 860, 1988, pp. 285-294.
7. R. Hughey and D. Lopresti, "B-SYS: A 470-Processor Programmable Systolic Array," *Proc. Int'l Conf. Parallel Processing*, IEEE CS Press, Los Alamitos, Calif., Order No. 2355-22, 1991, pp. 1580-1583.
8. M. Gokhale et al., "Building and Using a Highly Parallel Programmable Logic Array," *Computer*, Vol. 24, Jan. 1991, pp. 81-89.
9. H. Schroder and P. Strazdins, "Program Compression on the ISA," *Parallel Computing*, Vol. 17, No. 2-3, June 1991, pp. 207-215.
10. B. Friedlander, "Block Processing on a Programmable Systolic Array," *Proc. Int'l Conf. Parallel Processing*, IEEE CS Press, Los Alamitos, Calif., Order No. 889, 1987, pp. 184-187.
11. R. Smith and G. Sobelman, "Simulation-Based Design of Programmable Systolic Arrays," *Computer-Aided Design*, Vol. 2-3, No. 10, Dec. 1991, pp. 669-675.
12. S. Jones, A. Spray, and A. Ling, "A Flexible Building Block for the Construction of Processor Arrays," in *Systolic Array Processors*, Prentice-Hall, Englewood Cliffs, N.J., 1989, pp. 459-466.

Kurtis T. Johnson is an advanced engineer at HRB Systems, an E-Systems subsidiary. His interests include parallel computer architectures, parallel algorithms, and digital signal processing. He received his BS in 1986 and his MS in 1992, both in electrical engineering, from Pennsylvania State University.



A.R. Hurson is an associate professor of computer engineering at Pennsylvania State University. His research is directed toward the design and analysis of general-purpose and special-purpose computer architectures. He has published more than 110 papers on topics including computer architecture, parallel processing, database systems and machines, dataflow architectures, and VLSI algorithms. He coauthored the IEEE tutorial books, *Parallel Architectures for Database Systems* and *Multidatabase Systems: An Advanced Solution for Global Information Sharing Process*. He cofounded the IEEE Symposium on Parallel and Distributed Processing. Hurson is a member of the IEEE Computer Society Press Editorial Board and a member of the IEEE Distinguished Visitors Program.

Behrooz Shirazi is an associate professor of computer science engineering at the University of Texas at Arlington. Previously, he was an assistant professor at Southern Methodist University. His research interests include task scheduling, heterogeneous computing, dataflow computation, and parallel and distributed processing. He has published widely on these topics. He is a cofounder of the IEEE Symposium on Parallel and Distributed Processing. He has served as chair of the Computer Society section in the Dallas chapter of the IEEE and as chair of the IEEE Region V Area Activities Board. Shirazi received his MS and PhD degrees in computer science from the University of Oklahoma, in 1980 and 1985, respectively.

Send correspondence about this article to A.R. Hurson, Dept. of Computer Science and Engineering, Pennsylvania State University, University Park, PA 16802; or A2H@ecl.psu.edu.

Laxmi Bhuyan, *Computer's* system architecture area editor, coordinated and recommended this article for publication. His e-mail address is bhuyan@cs.tamu.edu.