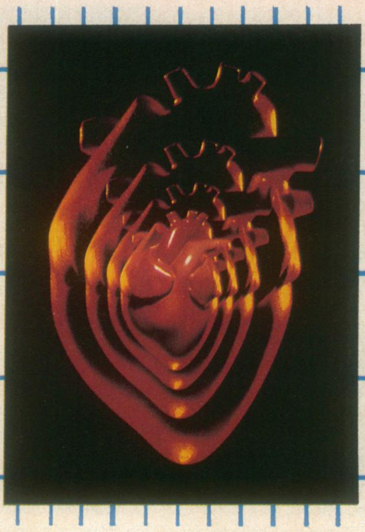

Guest Editors' Introduction

Systolic Arrays—From Concept to Implementation

José A.B. Fortes
Purdue University

Benjamin W. Wah
University of Illinois at Urbana-Champaign

Systolic arrays have regular and modular structures that match the computational requirements of many algorithms. Their implementation requires that a wealth of subsumed concepts and engineering solutions be mastered and understood.

Systolic arrays are the result of advances in semiconductor technology and of applications that require extensive throughput. Their realization requires human ingenuity combined with techniques and tools for algorithm development, architecture design, and hardware implementation.

Invariably, the first reaction of people who are exposed to the systolic-array concept is one of admiration for the concept's elegance and for its potential for high performance. However, those who next attempt to implement a systolic array for a specific application soon realize that a wealth of subsumed concepts and engineering solutions must be mastered and understood. This special issue attempts to provide insights into the implementation process and to illustrate the different techniques and theories that contribute to the design of systolic arrays.

Characteristics of systolic arrays

Since 1978, when H.T. Kung and C.E. Leiserson[1] introduced the term "systolic array" and the concept behind the term, much research has been done and much has been written about the design of algorithms and architectures suitable for such structures. Today, the idea of a systolic array is as familiar to many computer scientists and engineers as that of a compiler or a microprocessor.

The term "array" originates in the systolic array's resemblance to a grid in which each point corresponds to a processor and a line corresponds to a link between processors. As regards this structure, systolic arrays are descendants of array-like architectures such as iterative arrays,[2] cellular automata,[3] and processor arrays.[4] These architectures capitalize on regular and modular structures that match the computational requirements of many algorithms. Table 1 is a list of applications for which systolic designs are available. Systolic arrays belong to the generation of VLSI/WSI (Very Large Scale Integration/Wafer Scale Integration) architectures for which regularity and modularity are important to area-efficient layouts.

Although the array structure characterizes the interconnections in systolic arrays, it is the term "systolic" that captures the innovative and distinctive behavior of these systems. "Systolic" in this context means that pipelined computations take place along all dimensions of the array and result in very high computational throughput. In other words, systolic algorithms schedule computations in such a way that a data item is not only used when it is input but also is *reused* as it moves through the pipelines in the array. This results in balancing the processing and input/output bandwidths, especially in compute-bound problems that have more computations to be performed than they have inputs and outputs. Conventional processor designs are often limited by the mismatch of input bandwidth and output bandwidth, which occurs because data items are read/written every time they are referenced.

One reason for choosing "systolic" as part of the term "systolic array" was to draw an analogy with the human circulatory system, in which the heart sends and receives a large amount of blood as a result of the frequent and rhythmic pumping of small amounts of that fluid through the arteries and veins. In this analogy, the heart corresponds to a source and destination of data, such as a global memory, and the network of veins is equivalent to the array of processors and links. Another explanation of the term is that in many of the first proposed systolic architectures, processing elements alternated between cycles of "admission" and "expulsion" of data—much in the same way that the heart behaves with respect to the pumping of blood.

In the article "Why Systolic Architectures?"[5] H.T. Kung presents an excellent introduction to the basic ideas, the advantages, and the open problems of systolic arrays. Today, this article is still essential reading for those interested in learning the fundamentals of systolic arrays. Our introduction endeavors neither to replace nor to repeat the contents of that pioneering article. However, it is appropriate to elaborate briefly on the three factors that characterize systolic arrays as they were originally proposed, namely *technology, parallel/pipelined processing,* and *applications.* These factors also identify the reasons for the success of the concept, namely cost-effectiveness, high performance, and the abundance of applications for which systolic arrays can be used.

Technology and cost-effectiveness. Nowadays, mature VLSI/WSI technology permits the manufacture of circuits whose layouts have minimum feature sizes of 1 to 3 microns. The effective yields of VLSI/WSI fabrication processes make possible the implementation of circuits with up to half a million transistors at reasonable cost—even for relatively small production quantities. However, the advantages of this technology are not fully realized unless simple, regular, and modular layouts are used. Systolic arrays attempt to meet these topological constraints by using simple processing elements that, together with a simple interconnection pattern, are replicated along one or more dimensions. Cost, regularity, and modularity are factors leading to the design and optimization of individual processing elements and their respective interconnections. Consideration of these three factors indicates that processor arrays are cost-effective engineering solutions to the problem of building systems with many processing elements.

The main difference between the design of systolic arrays and that of other integrated systems of comparable complexity is illustrated in a general way in Figure 1. The Y-chart shown in the figure is a convenient and succinct description of the different phases of the process of designing VLSI systems.[6,7] The axes of the Y-chart correspond to orthogonal forms of system representation, and the arrows represent design procedures that translate one representation into another. A top-down design procedure (that is, one that progresses from more complex components to simpler subcomponents) can also be indicated—by arrows drawn along each axis and pointed toward the origin. While many different design approaches and—their corresponding Y-charts—are possible, design is typically carried out through *successive refinements.* In this process, a component's functional specification is translated first into a structural representation and then into a geometrical description in terms of smaller subcomponents; the functional description of each of these subcomponents must then be translated into structural and geometrical descriptions in terms of even smaller parts, and so on. The line arrows shown in the figure are intended to convey, in a general way, the flow of this process for systolic arrays versus more conventional systems. Since a systolic array consists of a large number of a few types of modules, the process of refining the overall system and designing every subcomponent is faster and simpler than it is in systems with the same size but a much larger number of module types. This is conveyed graphically in Figure 1 by means of large arrows showing that in the design of a systolic array, one can proceed faster and more directly to the design of lower-level components of the system than in traditional design.

Commercially available systolic-array chips with 10 to 100 simple, 1-bit proces-

Table 1. Applications for which systolic designs are available.

Signal and Image Processing and Pattern Recognition
FIR, IIR filtering, and 1D convolution
2D convolution and correlation
Discrete Fourier Transform
Interpolation
1D and 2D median filtering
Geometric warping
Feature extraction
Order statistics
Minimum-distance classification
Covariance matrix computation
Template matching
Seismic signal classification
Cluster analysis
Syntactic pattern recognition
Radar signal processing
Curve detection
Dynamic scene analysis
Image resampling
Scene matching
Matrix Arithmetic
Matrix-matrix multiplication
Matrix triangularization
QR decomposition
Sparse-matrix operations
Solution of triangular linear systems
Non-Numeric Applications
Data structures—stacks and queues, sorting
Graph algorithms—transitive closure, minimum spanning trees
Connected components
Language recognition
Dynamic programming
Arithmetic arrays
Relational database operations
Algebra

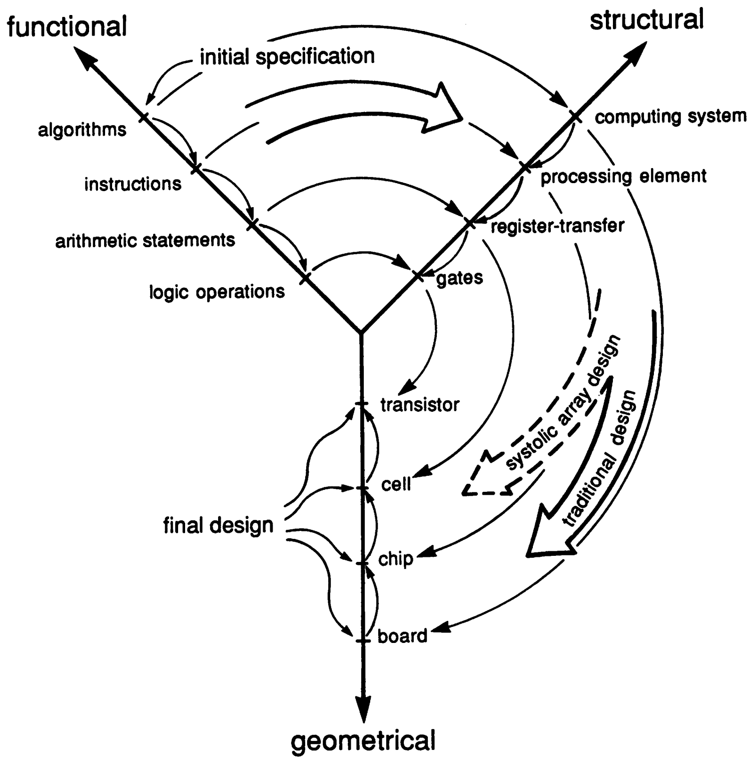

Figure 1. A Y-chart that shows the process of designing algorithmically specified VLSI digital systems.

sors exist; these chips sell for less than one hundred dollars apiece. Other chips, including microprocessors and digital-processing chips, both of which can be used as building blocks in systolic arrays, are also available—at even lower cost. Systolic arrays with thousands of processors can be built by assembling many such building blocks (chips) at total prices that range from ten thousand to a hundred thousand dollars and depend on the complexity of each processor.

Parallel/pipelined processing. Systolic arrays derive their computational efficiency from multiprocessing and pipelining. *Multiprocessing* is a natural consequence of the activities going on simultaneously in various processing elements of the array. *Pipelining* can be thought of as a form of multiprocessing that optimizes resource utilization and takes advantage of dependencies among computations. In systolic arrays, data pipelining reduces the input/output-bandwidth requirements by allowing a data item to be reused once it enters the array. Typically, inputs enter the array through peripheral processing elements and are propagated to neighboring processing elements for further processing. These movements of data through the array take place both along a fixed direction in which a link exists between neighboring processing elements and in a periodic manner.

In addition to data pipelining, systolic arrays are also characterized by *computational pipelining,* in which information flows from one processing element to another in a prespecified order. This information can be interpreted by the receiver as data, control, or a combination of both. Each output is computed by the execution—at different times and in a predetermined sequence—of several operations in a number of processing elements; the execution is performed in such a way that the output generated by one processing element is used as an input by a neighboring processing element. While operations can occur as data flows through each processor, the overall computation is not a dataflow computation, since the operations are executed according to a schedule determined by the systolic-array design. After a processing element generates an intermediate output and sends this output to the element's neighboring processing elements, the element computes another intermediate output. As a result, processing resources are utilized efficiently. In the general case, each processing element can be constructed as a pipelined processor. Such construction results in the so-called *two-level pipelined systolic array* and in even higher throughputs.

Applications and algorithms. Algorithms suitable for implementation in systolic arrays can be found in many applications, such as digital signal and image processing, linear algebra, pattern recognition, linear and dynamic programming, and graph problems. In fact, most of the algorithms in the listed applications are computationally intensive and require systolic architectures for their implementations when used in real-time environments. The acceptance of this fact is evidenced by the existence of prototype and production systolic arrays for modern real-time digital signal processing systems. The manufacturers of these arrays include, among others, companies such as ESL-TRW, Hughes, NCR, GE, Hazeltine, and Motorola. When systolic arrays were first proposed, they were intended for applications with two important sets of characteristics. First, these applications require high throughput and large processing bandwidth, possibly at the cost of increased response time. In other words, it is more important to keep up with the flow of data than to generate a set of outputs for a given set of inputs as quickly as possible. Second, these applications can be efficiently supported by algorithms that can be implemented on arrays consisting of a few types of simple processing elements; the arrays have simple controls and input/output ports in the peripheral processing elements. These algorithms are characterized by repeated computations of a few types of relatively simple operations that are common to many input data items. Often the algorithms can be described by programs with nested loops or by recurrence equations that describe computations performed on indexed data. In addition, the pattern of generation and usage of data by different operations displays some regularity and uniformity, which means that the resulting communication requirements can be met by the localized interconnections.

Implementation issues

Given the technical and economic principles that assure the soundness of the systolic-array concept, one needs to consider the issues involved in implementing a system for a specific application. Some of these issues are briefly discussed here.

General-purpose and special-purpose systolic systems. Typically, a systolic array can be thought of as an algorithmically specialized system in the sense that its design reflects the requirements of a specific algorithm. However, it may be desirable to design systolic arrays that are capable of efficiently executing more than one algorithm for one application or more. Two approaches are possible in designing these "large-purpose" systems, and a compromise between the two is often found in many actual implementations. One approach is based on adding hardware mechanisms so as to reconfigure the topology and interconnection pattern of the systolic array and to emulate the requirements of a specialized design. A concrete example of this approach is the Configurable Highly Parallel computer (CHiP),[8] which has a programmable lattice of switches for reconfiguration purposes. The other approach uses software to map different algorithms into a fixed-array architecture. As is the case with the approach behind other general-purpose parallel computers, this approach may require the use of programming languages capable of expressing parallel computations, as well as the development of translators, operating systems, and programming aids. These requirements apply, for example, in the case of Warp,[9] a systolic array developed at Carnegie Mellon University. For each algorithm, the designer needs to identify the efficient systolic designs and mappings and the appropriate techniques to use. The issue of appropriate techniques is of great importance, since the final performance, cost, and correctness of the design are governed by these techniques.

Design and mapping techniques. To synthesize a systolic array from the description of an algorithm, a designer needs a thorough understanding of and familiarity with the principles behind four things: systolic computing, the application, the algorithm, and the technology. Such skilled designers can provide excellent heuristic designs for important algorithms. However, the process is slow and error prone and may require extensive simulations, and the resulting designs are not guaranteed to be optimal or correct. Progress has been made in the development of systematic design techniques to automate this process.[10] These techniques are unlikely to replace the designers completely; instead, they will provide tools and formal concepts to assist designers in searching for diverse and desirable designs for a given application. Most of these techniques are concerned with the derivation of a relatively high-level specification of the array architecture from a description of the algorithm. Typically, such a specification includes the size and topology of the array, the operations performed by each processing element, the order and timing of data communication, and inputs and outputs. To a limited extent, these techniques can take into account technological factors and the relationship of the systolic array itself to the rest of the system. However, they are not complete; they can only be used at the specification level—and only in an indirect manner there. Until more is learned about design techniques that can be used conveniently for detailed integration of system and technology, such integration problems will continue to be left for the designer to solve.

Many specialized arrays can be seen as hardware implementations of a given algorithm.

Granularity. The basic operation performed in each cycle by each processing element in the various systolic arrays can range from a simple bit-wise operation, to word-level multiplication and addition, and even to execution of a complete program. The choice of granularity is determined by the application, or the technology, or both. For example, applications that use algorithms with basic bit-level operators and data structures naturally suggest that processing elements be of a corresponding complexity. The same choice of processing elements might, however, result from considerations such as input/output-pin restrictions and the technology that may be used. In programmable systolic arrays, the granularity may also be determined by trade-offs between the desired degree and level of programmability. The Saxpy Matrix-1[11] is an example of a programmable systolic computer with large granularity, whereas bit-level systolic arrays, like those discussed by J.V. McCanny and J.G. McWhirter,[6] are special-purpose designs with low granularity.

Extensibility. Many specialized systolic arrays can be regarded as hardware implementations of a given algorithm. This view holds when there is a direct correspondence between the operations and variables of the algorithm and, respectively, the processing elements and wire links of the systolic array. In such a case, the systolic processor can execute only a given algorithm that is designed for a problem of a specific size. If one wishes to execute the same algorithm for a problem of a larger size, then either a larger array must be built or the problem must be partitioned. The first approach is easy to conceptualize and simply requires that more processing elements be used to construct an enlarged version of the original array. However, as regards implementation, one must remember that there may be factors that do not affect performance in small arrays but might affect it in larger systems. These factors include clock synchronization, reliability, power requirements, chip-size limitations, and input/output-pin constraints.

Clock synchronization. In large synchronous systolic arrays, clock lines of different lengths can introduce clock skews and may require that a slower clock be used. Possible approaches that avoid this problem of clock skews include designing systolic arrays that do not allow data to flow in opposite directions and using efficient layouts of the clock distribution network.[12] An alternative to the design of a globally synchronous array is to achieve a self-timed system through the use of asynchronous handshaking mechanisms established between neighboring processing elements. These self-timed implementations are commonly referred to as *wavefront arrays.*[13]

Reliability. Simple laws of probability can be used to explain why increasingly large arrays are decreasingly reliable unless redundancy is incorporated and fault-tolerance mechanisms are available. In fact, the reliability of an array of processors is equal to that of a processor raised to a power of the number of processors in the array. Since the reliability of a processor is a value less than one, the reliability of the global array quickly approaches zero as the number of processors increases. Fault tolerance requires that faults be detected and located so that faulty processing elements can be replaced by operational spares through an appropriate reconfiguration scheme. A fault-tolerant systolic array may need additional hardware to meet these requirements. In addition, if time redundancy is used or system operation needs to be suspended for testing purposes, the fault-tolerant array can be slower than the original one. A good fault-tolerant design has as its goal maximizing reliability while minimizing the corresponding overhead. In systolic arrays, possible approaches to fault tolerance include simple extensions of well-known techniques used in conventional digital systems. However, these techniques do not take advantage of the characteristics of either systolic arrays or the algorithms they execute. Novel and successful, though general, fault-tolerance schemes[14] that take advantage of these characteristics have been proposed for systolic arrays.

Partitioning of large problems. When it is necessary to execute a large problem without building a large systolic array, the problem must be partitioned so that the same algorithm can be used to solve the smaller problem and so that an array of small, fixed size can be used. The main concerns are to avoid rendering the partitioned algorithm incorrect and to avoid increasing the complexity of the design significantly. One approach identifies algorithm partitions and an order of execution of these partitions such that correctness is preserved and the original array can be used to execute each partition.[15] The perceived result of this approach is that the array "travels" through the set of computations of the algorithm in the right order until it "covers" all the computations. Another approach attempts to restate the problem to be solved so that the problem becomes a collection of smaller problems that is similar to the original one and that can be solved by the given systolic array.[16] While this second approach has less generality and is harder to automate than the first approach, it may have better performance when it is applicable.

Automated design tools. The processing elements and module libraries play an important role in making the process of designing special-purpose arrays of processing elements faster and more cost-effective. In addition to the many existing tools for designing VLSI and WSI systems that can be readily used in this process, the regularity and algorithmic nature of systolic arrays permits the use of high-level silicon compilers.[7] At this time, the development process is not fully automated; the process will depend on future progress in design automation and computer-aided design tools.

Universal building blocks. Systolic arrays cost less to implement than other arrays because of their extensive replication of a small number of simple, basic modules and because of their highly dense and efficient layouts. It is worthwhile for the simple building blocks to be carefully designed and optimized, since the costs involved are amortized over a large number of replicated circuits. The modular design of systolic arrays allows designers who want rapid prototyping of their ideas to use off-the-shelf devices, such as microprocessors, floating-point arithmetic units, and memory chips. However, these parts may not be designed for implementing systolic arrays and may therefore be inadequate to meet the design requirements. This has led to the development of "universal building blocks"—chips that can be used for many systolic arrays. The cost of such development is, therefore, amortized over replicated modules in many arrays rather than concentrated in simply one array. Commercially available chips that are worthy of consideration as basic modules include the INMOS Transputer, the TI TMS32010 and TMS32020, the NEC dataflow chip μPD 7281, Analog Devices' ADSP2100, the Fujitsu MB8764, and the National LM32900. Problems involved in the use of programmable building blocks include developing programming tools to aid designers and providing support for flexible interconnections.

Integrating systolic arrays into existing systems may be nontrivial because of I/O bandwidth.

Integration into existing systems. Although systolic arrays provide extensive throughput, their integration into existing systems may be nontrivial because of the extensive input/output bandwidth involved, especially when a problem has to be partitioned and input data have to be accessed repeatedly. Additional problems that have to be solved for systems with a large number of systolic arrays include the interconnections with the host, the memory subsystem to support the systolic arrays, the buffering and access of data to meet the special input/output data distributions, and the multiplexing and demultiplexing of data when there are insufficient input/output ports. The problems that must be faced are exemplified by Mosaic,[17] a project being carried out at ESL. The system consists of a statically scheduled crossbar switch that connects multiple Warp processors, each with local memory modules, into a macropipeline. The local memory modules are used to store input data and restructure them into the required input format.

The future

By the year 2000, it will be possible to build integrated circuits with one billion transistors—more than one thousand times the number of devices available in today's densest integrated circuits.[18] These incredibly large circuits will use 0.1—micron geometries made possible by advanced optical, electron-beam, ion-beam, or X-ray lithography. While the high cost of setting up integrated-circuit factories that can handle these technologies will certainly impact the initial cost per chip, the main manufacturing limitations will be in the design, verification, testing, and packaging of such large circuits. In addition, the percentage of the chip area dedicated to interconnections could increase to more than 80 percent. Systolic arrays will take advantage of submicron technologies without suffering from the problems just mentioned, since they are modular, have regular interconnections, and are extensible. By the year 2000, mature design and programming tools and extensive knowledge of suitable applications and algorithms will probably render systolic arrays the architecture of choice for submicron circuits designed for digital signal processing, fast arithmetic, symbolic processing, and intelligent databases.

Systolic arrays have triggered extensive related work and research in the areas of processor-array architecture, algorithm design and analysis, and parallel programming. These areas are often identified as *systolic architecture, systolic algorithms,* and *systolic computing,* respectively. As a consequence, the principles behind systolic arrays have gained an enlarged scope. That is, systolic architectures are not necessarily arrays of processors; systolic algorithms may be very complex and may not necessarily be executed in simple processing elements; and systolic computing can take place in computers other than systolic architectures. The prominent features of systolic arrays are the processing elements, which implement processes, and the regular interconnection of multiple processing elements. The processing elements and the interconnection of processing elements can be implemented in software, general-purpose microprocessors, or specialized hardware. Because of this variety of implementation possibilities, systolic arrays have, since the late seventies, evolved to become cellular computing at the algorithmic, programming, architectural, and hardware levels. We are, therefore, witnessing a trend in which systolic computing is becoming a pervasive form of multiprocessing. □

Acknowledgments

We would like to thank the authors and reviewers for helping to make this special issue a reality. We are also grateful to Bruce Shriver, the editor-in-chief of *Computer,* for his guidance, directions, and help in preparing this special issue.

Despite the large number of articles in this special issue and despite our efforts to solicit manuscripts on major systolic-array projects for it, we were not able to cover all major projects because of page limitations and the tight schedule involved in preparing the issue. We realize that there are many researchers, too numerous to mention individually, who have made notable contributions to the development of systolic-array research. We apologize for any inadvertent omissions, and we would like to acknowledge their efforts here.

Last but not least, we would like to acknowledge the pioneering study on systolic arrays by H.T. Kung and C.E. Leiserson. Without their study, this special issue would not exist.

This project was supported by National Science Foundation Grants DCI 84-19745 and DCI 85-19649, as well as by the Innovative Science and Technology Office of the Strategic Defense Initiative Organization under Office of Naval Research Grant 00014-85-k-0588.

References

1. H.T. Kung and C.E. Leiserson, "Systolic Arrays (for VLSI)," *Sparse Matrix Proc. 1978,* 1979, Academic Press, Orlando, Fla., pp. 256-282; also in "Algorithms for VLSI Processor Arrays," which is Section 8.3 of *Introduction to VLSI Systems,* C. Mead and L. Conway, eds., 1980, Addison-Wesley, Reading, Mass., pp. 271-292.
2. F.C. Hennie, *Iterative Arrays of Logical Circuits,* 1961, MIT Press, Cambridge, Mass.
3. J. von Neumann, "The General Logical Theory of Automata," in *Cerebral Mechanisms in Behavior—The Hixon Symposium,* L.A. Jeffries, ed., 1951, John Wiley & Sons, New York; a more recent work on cellular automata is *Modern Cellular Automata Theory and Applications* by K. Preston, Jr., and M.J.B. Duff, 1984, Plenum Press, New York.
4. D.L. Slotnick, W.C. Borck, and R.C. McReynolds, "The Solomon Computer," *Proc. AFIPS Fall Joint Computer Conf.,* 1962, Spartan Books, Washington, DC, pp. 97-107.
5. H.T. Kung, "Why Systolic Architectures?" *Computer,* Vol. 15, No. 1, Jan. 1982, pp. 37-46.
6. J.V. McCanny and J.G. McWhirter, "Some Systolic Array Developments in the United Kingdom," *Computer,* Vol. 20, No. 7, July 1987 (this issue).
7. *Computer* (special issue on new VLSI tools), Vol. 16, No. 12, Dec. 1983.
8. L. Snyder, "Introduction to the Configurable, Highly Parallel Computer," *Computer,* Vol. 15, No. 1, Jan. 1982, pp. 47-64.
9. M. Annaratone et al., "Warp Architecture and Implementation," *Proc. 13th Int'l Symp. Computer Architecture,* June 1986, Computer Society Press, Silver Spring, Md., pp. 346-356.
10. J.A.B. Fortes, K.S. Fu, and B.W. Wah, "Systematic Approaches to the Design of Algorithmically Specified Systolic Arrays," *Proc. 1985 Int'l Conf. Acoustics, Speech, and Signal Processing,* 1985, IEEE, Piscataway, N.J., pp. 8.9.1-8.9.5.
11. D.E. Foulser and R. Schreiber, "The Saxpy Matrix-1: A General-Purpose Systolic Computer," *Computer,* Vol. 20, No. 7, July 1987 (this issue).
12. A.L. Fisher and H.T. Kung, "Synchronizing Large VLSI Processor Arrays," *IEEE Trans. Computers,* Vol. C-34, No. 8, Aug. 1985, pp. 734-740.
13. S.Y. Kung et al., "Wavefront Array Processors: From Concept to Implementation," *Computer,* Vol. 20, No. 7, July 1987 (this issue).
14. J.A. Abraham et al., "Fault Tolerance Techniques for Systolic Arrays," *Computer,* Vol. 20, No. 7, July 1987 (this issue).
15. D.I. Moldovan and J.A.B. Fortes, "Partitioning and Mapping Algorithms Into Fixed-Size Systolic Arrays," *IEEE Trans. Computers,* Vol. C-35, No.1, Jan. 1986, pp. 1-12.
16. J.J. Navarro, J.M. Llaberia, and M. Valero, "Partitioning: An Essential Step in Mapping Algorithms Into Systolic Array Processors," *Computer,* Vol. 20, No. 7, July 1987 (this issue).
17. F.C. Lin et al., "MOSAIC: A Heterogeneous Architecture for Signal Processors," *Proc. 12th DARPA Strategic Systems Symp.,* Oct. 1986.
18. B.C. Cole, "Here Comes the Billion-Transistor IC," *Electronics,* Vol. 60, No. 7, Apr. 2, 1987, pp. 81-85.

José A.B. Fortes has been with the faculty of Purdue University's School of Electrical Engineering since 1984.

He is interested in all aspects of parallel processing, including the systematic design of algorithmically specialized processor-array architectures, parallel programming languages, automatic parallelism detection and exploitation techniques, and fault-tolerant computing.

Fortes has published over 20 technical papers in journals and conference proceedings in the areas of parallel processing, fault-tolerant computing, and VLSI architectures. He has worked on several projects in these areas in cooperation with or with funding from NSF, ONR, AT&T, RCA, and NCR.

Fortes is a member of IEEE and ACM.

He received his MSEE and PhD EE degrees from Colorado State University and the University of Southern California in 1981 and 1983, respectively.

Benjamin W. Wah is an associate professor in the Dept. of Electrical and Computer Engineering and in the Coordinated Science Laboratory of the University of Illinois at Urbana-Champaign.

He was on the faculty of the School of Electrical Engineering at Purdue University between 1979 and 1985.

His current research activities include parallel computer architectures, artificial intelligence, distributed databases, computer networks, and theory of algorithms.

Wah was a Computer Society Distinguished Visitor between 1983 and 1986.

He is an editor of the *IEEE Transactions on Software Engineering,* and the *Journal of Parallel and Distributed Computing.*

He received the PhD in computer science from the University of California at Berkeley in 1979.

Readers may write for information about this special issue to José Fortes, Purdue University, School of Electrical Engineering, West Lafayette, IN 47907 or to Benjamin Wah, University of Illinois at Urbana-Champaign, Coordinated Science Laboratory, 1101 W. Springfield Ave., Urbana IL 61801.