from IEEE Computer, July 1987, v. 20, no. 7, pp. 91-103
also in Selected Reprints from Computer Society Magazines, IEEE, 1987
pp. 1-13

Systolic Arrays

A Survey of Seven Projects

José A.B. Fortes
Purdue University, West Lafayette, IN 47907

Benjamin W. Wah
University of Illinois at Urbana-Champaign, Urbana, IL 61801

This special section provides a concise overview of seven projects concerned with the design and implementation of systolic arrays.

The issue of *Computer* that you are now reading showcases six research projects; the feature articles in which they are described all discuss the realization of systolic arrays from concept to implementation. This special section provides an overview of seven more projects that have to do with the design and implementation of systolic arrays. The first six projects are concerned with the implementation of signal/image-processing applications. The last project is concerned with the design of hexagonally connected processing elements in such a way as to allow the direct execution of dataflow graphs. These seven projects show a variety of successful but different implementations of application-dependent systolic arrays.

The collection of research projects described in this section is listed below, together with the entities that are carrying out the projects. These include

- a GaAs systolic array for adaptive null steering beamforming, RCA;
- a systolic processor for adaptive beamforming, ESL;
- a systolic array for digital signal processing, Motorola;
- a systolic array for linear algebraic and cellular operations in signal processing, Hughes Research;
- P-NAC, a systolic array for comparing nucleic acid sequences, Brown University;
- Linc, a reconfigurable systolic array for signal processing, General Electric; and
- a hexagonally connected processing array for mapping dataflow graphs, Technion (Israel) and the University of Massachusetts.

Each summary includes the name and address of at least one person who can be contacted for further information. The summaries also reference published work in the field.

We hope that this special section will serve as the beginning of a forum for the exchange of ideas.

The Design of a GaAs Systolic Array for an Adaptive Null Steering Beamforming Controller

Carl E. Hein, Richard M. Zieger, and Joseph A. Urbano
RCA Advanced Technology Labs, Moorestown, NJ 08057

Our design of the RCA GaAs systolic array beamforming controller demonstrates the advantages of using a top-down approach for designing an adaptive radar system. Because of its speed and unique characteristics, the GaAs technology chosen to implement the array significantly influenced the design of the internal processor architecture. The array is configured as an SIMD machine in which each processing site communicates only with its nearest neighbors. The array is intended for use in digital radar beamforming for automatic null steering.

Several systolic arrays have been proposed or constructed for similar applications. Among them are the RCA Gram-Schmidt Preprocessor,[1] the NOSC (Naval Ocean Systems Center) systolic array,[2] and the Carnegie Mellon Warp[3] machine, which was constructed in cooperation with General Electric and Honeywell.

The selection of an algorithm that could adaptively generate the complex beamforming coefficient vector was the first step in our design process. Of the many such algorithms studied, including MSR3, Gram-Schmidt, and Givens, the MSR3 Algorithm was identified as offering the best combination of performance, stability, and ease of systolic implementation. Each algorithm's performance was judged by the algorithm's speed of convergence, the depth of nulls, and the shape of the antenna pattern produced. The MSR3 Algorithm became the design guide for the systolic array.

The MSR3 Algorithm iteratively converges on a solution to the beamforming coefficients by minimizing the error in the energy received by the antenna. The computational core of each iteration consists of a pair of doubly nested DO loops; each loop has several arithmetic operations inside. Such a sequential code structure implies a two-dimensional systolic array of processors.

For a doubly nested loop, a two-dimensional array of processing nodes can be used in which each node corresponds to an iteration of the outer loop, and each column corresponds to an iteration of the inner loop. In the application, the inner-array index limit is dependent on the outer-array index, as shown in the following:

```
FOR i := 1 to N DO
  FOR j := 1 to i DO
     X(i + 1,  j) := X(i,j) * Y(i,j);
     Y(i,  j + 1) := X(i,j) * Y(i,j);
FOR i := N to 1 DO
  FOR J := i to 1 DO
     X(i + 1,  j) := X(i,j) * Y(i,j);
     Y(i,  j + 1) := X(i,j) * Y(i,j);
```

This dependency eliminates half of the node positions and results in a triangular array.

A doubly nested loop can be implemented as a linear array in which each node performs every iteration of the inner loop for each iteration of the outer loop. However, implementing this outer loop would result in an unbalanced processing load because, for each invocation, the $i = 1$ processing node has only one iteration to perform, whereas the $i = N$ processing node has N iterations.

The simulation of the algorithm does not contain assumptions or information about the physical structure of the computing elements. As a result, the simulation computes the algorithm by performing each operation in sequence. To assist in determining the order of data transfers and operations in the full array of processors running under the steady-state condition, we modified the original simulation of the algorithm into a system simulation that recognized the parallel execution of the array and the distribution of variables in space and time.

From the system simulation, the sequence and locations of the data transfers between nodes were easily traceable. Of the algorithm's two pairs of DO loops, the second pair has a deleterious effect on the performance of the array because it is indexed in reverse order from the first. In the systolic array, dataflow in the first DO-loop pair flows from left to right and from top to bottom, but in the second DO-loop pair, data flows from right to left and bottom to top. The implication is that instead of the data from succeeding invocations propagating in continuous waves across the array, one wave must propagate across the array in one direction, then another wave must propagate across the array in the opposite direction. This destroys the ability to pipeline the operation efficiently and results in the majority of nodes being idle most of the time.

We chose a unique combination of system and nodal architecture; this architecture produces continuous, pipelined dataflow by providing storage locations at each node that allow both waves to run simultaneously in opposite directions on one triangular array of processors. It does this by creating a pipeline of operations that has a depth equal to twice the number of processors along the diagonal of the array. The storage registers accommodate the numerous stages of the pipeline by saving the critical values from each stage. In this way, the dataflow of the algorithm is preserved in space because it skews operations over time.

The intricate data and time dependencies within the systolic array are so complicated that it was virtually impossible to continue the design without simulation at the system level. The system simulation was useful in determining bottlenecks and critical paths at the system level; it employed information about the length of time required, and about the data produced and consumed, by each operation. Initially, these quantities were estimated. As the design progressed and more information became available, the simulation determined more precisely the time constraints for each operation and data transfer.

Simulation demonstrated that under steady-state processing, the processors in the array run a single sequence of operations in three phases with a 100 percent utilization of the interior processors.

The system architecture of a systolic array can be made fault tolerant if a nodal interconnection scheme is provided so that a failed processing node can be bypassed and processing nodes in spare rows and columns can be used.

The system architecture set certain constraints on the internal architecture of the nodal processor and dictated the processor's functionality. Communication ports going in and out are required on each of the four sides of the nodal processor. Both the simplicity of the operations and the construction advantages suggested that all circuitry for a processing node can be partitioned onto one chip. The data ports were serialized by nibbles to lessen their number. The algorithm in combination with the system architecture dictated the nodal processor's register set, which consists of four general registers used for holding critical values of the algorithm between operations.

A throughput requirement on the order of 5 μs per coefficient update indicated the need for clock rates in the 100-MHz range. These clock rates are currently reachable with GaAs parts. Unfortunately, GaAs fabrication capabilities are not as advanced as those of silicon. Consequently, the internal architecture of the nodal processor reflects many trade-offs in circuit complexity. Since the processor is being custom built to perform MSR3-like algorithms, only the minimum set of data paths is provided, and control is effected through logic driven by a state counter.

GaAs offers many advantages.

Circuit complexity was further reduced 25 percent by the selection of a special pseudo-floating-point number format that offers the equivalent range of 14-bit fixed-point representation but uses only 10 bits. This format was developed to meet range and accuracy requirements of the beamforming application. This format reduces the interchip communication volume by 29 percent.

Gallium arsenide technology offers important advantages for real-time signal-processing applications. GaAs has a high speed/power factor, is radiation hard (in terms of total dose), and temperature tolerant. Although GaAs has problems that arise from its limited radiation hardness, high wafer density dislocations (which result in low yield), high cost, small noise margin, and the lack of high-speed testing equipment available for it, most GaAs problems are expected to be temporary.

For the GaAs systolic array, RCA has designed and fabricated a GaAs 32-bit ALU test chip and a 500M-bps Manchester Encoder. A GaAs 200-MIPS, RISC, 8-bit microprocessor and a Manchester Decoder were also designed. The GaAs chips were fabricated by Triquint Corp. of Beaverton, Ore., by means of a low-power process (a typical gate dissipates about 780 μW) that incorporates 1-micron enhancement/depletion MESFET technology in both gate-array and standard-cell options.

The standard cells used in the systolic array are limited to high-yield 1- to 5-input NOR gates. The systolic array includes single-chip, standard-cell, GaAs processing nodes. Each processing-node chip has about 2100 of the small NOR gates.

Working from specifications of data-transfer rate, coefficient-update rate, and GaAs technology limitations (as regards gates and pins), we determined the minimum clock rate, the number and types of chips, and the number of handshaking lines required.

In the array, only one processor design was used. The clock rate is 120 MHz; the rate of data transfer between processors is 24 MHz; the number of pins is 42. The system updates the coefficients for the multiple beams every five μs.

Data is transmitted serially over eight lines (that is, over the two unidirectional lines on each of four sides of a processing node in the array). A RISC-like philosophy of hardware-to-software trade-offs was used to meet the low-gate-count restriction.

The efficient design procedure reported here has resulted in the design of a realizable, special-purpose, hardware-efficient systolic array with real-time performance unmatchable by any realizable uniprocessor system.

References

1. S.M. Yuen, "Algorithm and Systolic Architecture for Solving Gram-Schmidt Orthogonalization Systems," RCA/GSD and MSR, Moorestown, N.J.
2. Conversation held Jan. 13, 1987, with J. Loughlin of the Naval Ocean Systems Center, San Diego, Calif.
3. H.T. Kung, "A Programmable Systolic Array Machine and Its Applications," presentation on parallel architectures for signal processing, given at an IEEE Philadelphia Section meeting held at the University of Pennsylvania, Philadelphia, in Apr. 1986.

A Systolic Signal Processor for Signal-Processing Applications

Douglas A. Kandle
ESL, Inc., 495 Java Dr., MS 302, PO Box 3510, Sunnyvale, CA 95088

As part of its Adaptive Beamformer project, ESL, Inc., has completed a 350-MFLOP systolic processor that performs adaptive beamforming for acoustic signal-processing applications. The adaptive beamformer itself implements the Minimum Variance Distortionless Response Algorithm.[1] A frequency-domain adaptive beamformer, it was developed on a systolic architecture processor implemented with custom VLSI chips. ESL's Systolic Adaptive Beamformer project is intended to demonstrate the applicability of systolic processing techniques to acoustic signal processing.

The system consists of a special-purpose systolic processor that attaches to a sensor array (100 channels, each of which produces 12-bit data samples at a rate of 3000 samples/sensor/sec) and to a general-purpose host computer (the Digital Equipment Corp. VAX 11/750). Figure 1 is a block diagram of the system.

The processor was built to perform narrow-band, passive, sonar signal processing. The basic signal-processing requires that the inputs from many hydrophones be linearly combined in the frequency domain so as to produce a directional acoustic receiver with very high gain. A sensor array has a complex spatial-response pattern. It will "hear" sounds from every direction; these have varying degrees of amplification (or attenuation). The goal of an adaptive beamformer is to combine the sensor outputs in such a way that the array will hear sounds coming from the desired direction but will minimize the total energy (from noise and interfering sources) received from all other directions.

A complete description of the adaptive beamforming process is beyond the scope of this article. However, from a computing-architecture point of view, the process can be broken down into three linear algebra problems that must be solved.

(1) Factor a matrix. (That is, compute a matrix $\mathbf{U}_{new}$ defined by $\mathbf{U}_{new}\mathbf{U}_{new}^{*} = \mathbf{U}_{old}\mathbf{U}_{old}^{*} + \mathbf{z}\mathbf{z}^{*}$. $\mathbf{U}_{old}$ is an $n \times n$ upper triangular matrix with real, positive values on the diagonal, and $\mathbf{z}$ is a vector in C^n.

(2) Solve linear systems of the form $\mathbf{U}\mathbf{x} = \mathbf{d}$, where $\mathbf{x}$ and $\mathbf{d}$ are vectors in C^n and $\mathbf{U}$ is as defined above.

(3) Compute inner products of the form $\mathbf{w}^{*}\mathbf{z}$ where $\mathbf{w}$ and $\mathbf{z}$ are in C^n.

ESL has developed a custom VLSI chip that is a systolic cell. The systolic-cell chip is used to solve all of the above problem types. The systolic-cell chip is a high-speed, floating-point multiplier/adder designed to support complex arithmetic. Chip operands are 32-bit, IEEE-format, floating-point numbers. Successive operands are treated as a sequence of real, then imaginary parts of complex numbers. The chip has three input-data ports (A, B, and C) and two output-data ports (BD and CD). All five data ports are 1 byte (8 bits) wide and require eight consecutive 1-byte data transfers to complete a complex operand transfer. Data moves at a rate of 1 byte every 100 ns, thus requiring 800 ns to transfer a complex value.

The principal operation performed by the systolic-cell chip is to accept three input operands (A, B, and C) and produce two results (BD and CD) such that

$$BD \leftarrow B$$
$$CD \leftarrow A \times B + C$$

The signals BD and CD are delayed seven cycles, which is the pipe depth of the chip.

In addition to the input-data ports, there are OC, FC, and BC ports. These ports

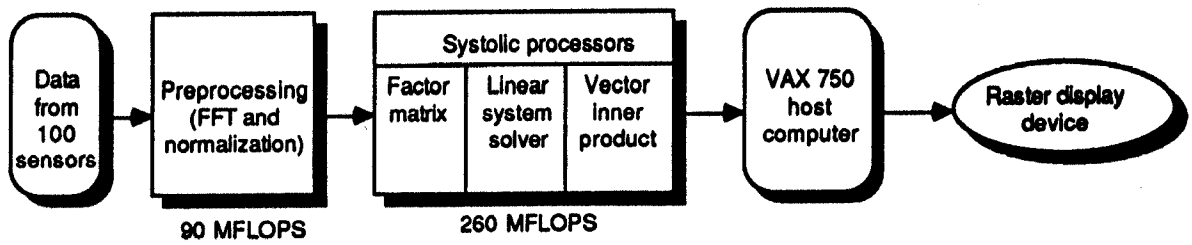


Figure 1. A block diagram of ESL, Inc.'s 350-MFLOP systolic processor system.

determine the operation performed and effectively constitute an "instruction." Inputs CA, CB, and CC selectively negate and/or conjugate the A, B, and C inputs.

The Systolic Adaptive Beamformer project has shown that a systolic-architecture processor can achieve very high processing throughput even in a very small physical space. Single-board systolic processors with a computational throughput of up to 120 MFLOPS were built. The 350-MFLOP system has achieved very high processing utilization rates (in excess of 90 percent of available computational power). To achieve its very high sustained-throughput rates, the system is tailored to a specific set of linear algebra problems. While this tailoring is very restrictive from a mathematical viewpoint, the class of problems that the systolic processors can solve is quite large. In general, the system can solve any linear least squares problem of order 50 or less. The upper bound of 50 is imposed by the memory sizes of the systolic processors.

ESL is currently building a second-generation systolic processor that will be able to perform at a substantially higher throughput. The original design has been improved by adding both more systolic cells in each processor and more flexible microcode addressing modes.

Acknowledgment

This work was funded by DARPA under Contract N00014-83-C-0110.

References

1.N.L. Owsley, "Sonar Array Processing," in *Array Signal Processing*, S. Haykin, ed., 1985, Prentice-Hall, Englewood Cliffs, N.J.

Suggested reading

Golub, G.H., and C.V. Van Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, Md., 1983.

Kandle, D.A., "A One-Third Gigaflop Systolic Adaptive Beamformer," *Proc. SPIE*, Vol. 433, 1985, SPIE, Bellingham, Wash.

Monzingo, R.A., and T.W. Miller, *Introduction to Adaptive Arrays*, 1980, John Wiley & Sons, New York.

An Advanced DSP Systolic-Array Architecture

Steven B. Leeland
Motorola, Inc., Government Electronics Group, 2501 S. Price Rd., Chandler, AZ 85248-2899

Motorola has been working on an advanced digital signal processing (DSP) systolic-array architecture. Progress to date includes a completed architecture design, a design for processing-element logic, architecture and algorithm simulations, and partially completed logic simulations. The primary goals for this architecture are

- to increase the processing performance by a factor of 16 over that of an existing Motorola DSP system (a custom VLSI) that executes demodulation algorithms,
- to perform 32-bit floating-point arithmetic for applications requiring great precision, such as a 64K point Fast Fourier Transform, and
- to reduce the effort needed to develop software for the architecture.

Processing-element size is a critical factor in a systolic-array architecture. Small processing elements allow more elements per chip, which in turn reduces overall system size. To reduce processing element size, Motorola has based the architecture on processing elements with 32-bit floating-point serial processors; the processing-element structure is shown in Figure 1. Serial processors require an order of magnitude less logic than parallel versions. The processors for the new architecture are serial, and they require 50 clock cycles to perform each operation. Chip operating speed should exceed 20 MHz, since the longest logic path is the one used by an 8-bit exponent adder. Therefore, a new data sample can be introduced into the array every 2.5 μs, which exceeds the rate for introducing new data samples in the existing Motorola DSP system by a factor of 20.

Because the floating-point processing consumes only 50 clock cycles, it is possible to improve intercell communications over those in the existing DSP. During

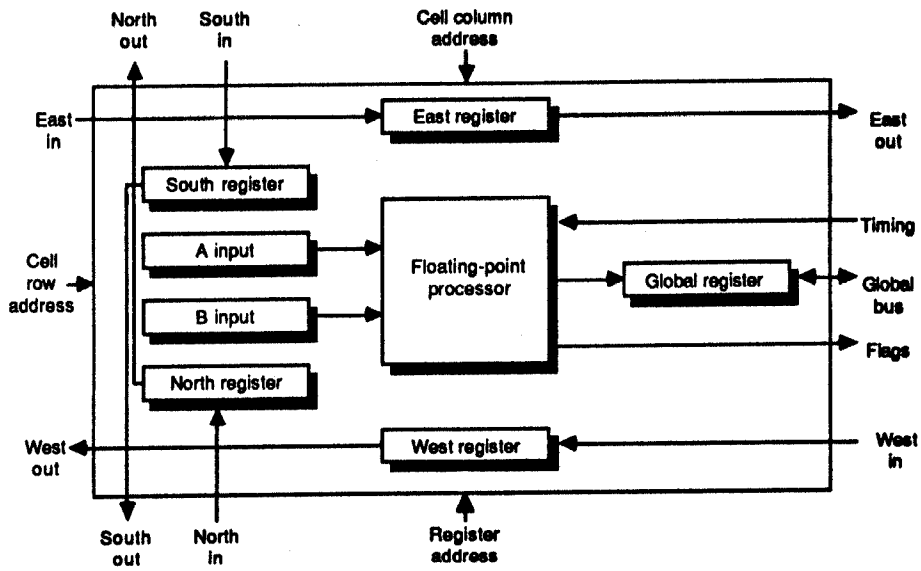


Figure 1. Processing element structure.

these normally unusable times (the clock cycles), data is allowed to flow through cells transparently. A network of East, West, North, and South buses allows data to travel as far as 50 cells away. Many systolic-array architectures only allow direct data transfer between nearest-neighbor processing elements.[1,2]

DSP-software development is often a time-consuming and arduous task. Motorola's new systolic-array architecture has already reduced this effort. Algorithms are implemented directly in this architecture. Every cell is initialized at least once after each power-up. From that point (power-up) on, the cell performs the same operation every processing cycle. It also gets its input data from the output of an assigned "neighbor" cell. This pre-set operation makes it possible to assign the function of each cell directly from a signal-flow diagram. The new systolic-array architecture essentially eliminates sequential software. The software-development tool is written in Pascal for an IBM PC. The basic concept for the development tool is a spreadsheet processor. Algorithms are entered, copied, and replicated just as in any spreadsheet tool. As an added feature, the algorithm can be tested, simulated, and debugged with the same tool.

Architecture simulations have reconfirmed that this systolic design performs best on algorithms with strong locality of signal flow.[3,4] The date when the architecture will be usable is drawing nearer as logic simulations approach completion.

The Systolic/Cellular System for Signal Processing

J. Greg Nash, K. Wojtek Przytula, and S. Hansen
Hughes Research Laboratories, 3011 Malibu Canyon Rd., Malibu, CA 90265

We designed the Systolic/Cellular System for large classes of linear algebraic and cellular operations that are used in signal processing. It consists of a host and a programmable coprocessor. The coprocessor includes an array of 16 × 16 mesh-connected processors, dual-port array memory, and a controller with a separate program memory (see Figure 1). The input data and the programs for the coprocessor are loaded from the host into the array memory and the program memory, respectively.

The system can operate in two modes: *cellular* and *systolic*. In the cellular mode of operation, the input data are first

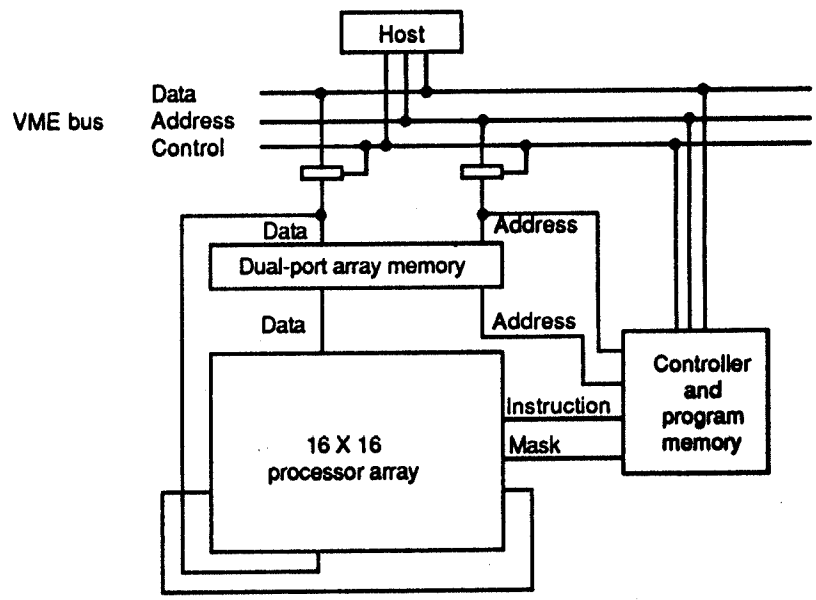


Figure 1. Block diagram of the Systolic/Cellular System.

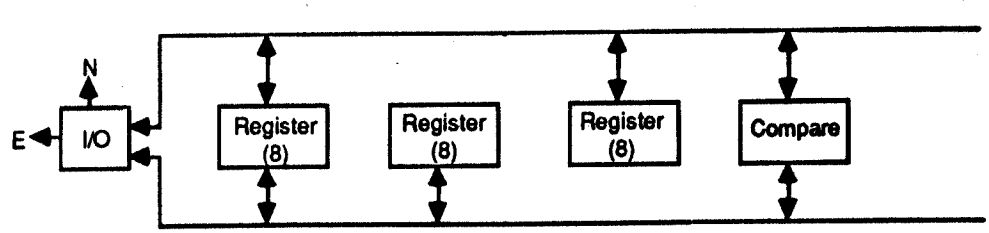


Figure 2. Organization of the processors.

References

1. "What's New in ICs," *Integrated Circuits Magazine*, Vol. 3, No. 1, Jan. 1985, pp. 68-70.
2. R. Davis and D. Thomas, "Systolic Architecture Tackles Image Feature Extraction," *Integrated Circuits Magazine*, Vol. 3, No. 3, Mar. 1985, pp. 28-37.
3. C. Mead and L. Conway, *Introduction to VLSI Systems*, Addison-Wesley, Reading, Mass., 1980.
4. S.Y. Kung, "On Supercomputing With Systolic/Wavefront Array Processors," *Proc. IEEE*, Vol. 72, No. 7, 1984, pp. 867-884.

loaded into the processor array from the array memory and are then processed in unison. After the computation is completed, the results are unloaded from the processors and the next computation cycle begins. We used this mode to implement point and window operations (thresholding, convolution, and so on), fast transforms (Haar, Hadamard, 1D and 2D FFT—each comes in radix 2 and 4 versions), sorting, and some of the matrix operations (addition, transposition).

In systolic computations, the data blocks leave the array memory in a row-by-row fashion through one port, then "flow" through the array, during which time they undergo appropriate processing. The results return to the array memory through the other port. The systolic operations are implemented by means of only two algorithms, both generic: the modified Faddeeva Algorithm[1] and the algorithm of Frank Luk.[2] We used the Faddeeva Algorithm for implementation of matrix operations (inversion, QR factorization), solution of systems of linear equations (including dense and banded systems larger than the array), and least squares problems (overdetermined, underdetermined, generalized). We used the algorithm of Frank Luk for matrix, eigenvalue, and singular-value decompositions. All the operations listed above are used frequently, not only in signal processing but also in many other applications—for example, in robotics.

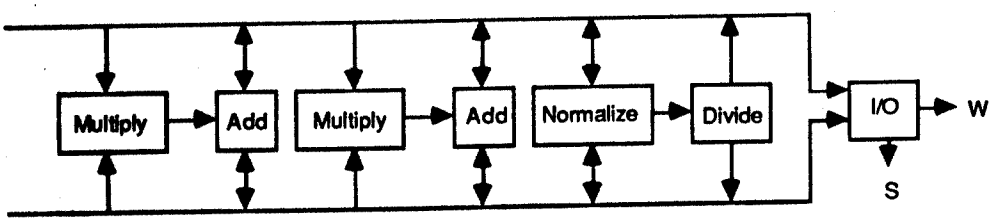
The architecture of the coprocessor is tailored to the two generic systolic algorithms, which have a similar underlying structure, and to selected cellular algorithms, such as fast transforms.[3] The processing is performed under the control of a simple SIMD controller. The leftmost column of the array (which consists of boundary processors) can be programmed to perform different operations from the rest of the processors (which are the internal processors). All processors within these two parts of the array operate in unison. However, it is possible to disable selected processors for any computation step.

A special, custom VLSI processor has been designed for the array (see Figure 2). It is a 32-bit, fixed-point, dual-bus processor with a bit-slice structure. There are seven functional units in it: two multipliers, two adders, a divider, and a comparator.[4] Each processor also contains 24 memory registers. The arithmetic algorithms of the functional units have the necessary control embedded in hardware, which adds considerable speed over that attained by means of a microcoded approach; also, multiple functional units can perform computations concurrently. Our choice of the functional units was dictated by the target applications. For example, FFT and Faddeeva algorithms require four multiplications per computation stage; therefore, we provided more than one multiplier. The processors have four multiplexed 16-bit I/O ports for interconnection with their four nearest neighbors. The control over the ports is separate from the control of the data path so that communication between the processing elements can take place during the computational activity.

Maximum system performance is in the neighborhood of 450 MOPS.

An instruction set of the array consists of about 30 powerful, wide instructions. The machine instruction, which is 112 bits wide, has two separate fields for operation codes for boundary and internal processors; additionally, these two fields are subdivided into I/O and computation subfields. A third field is used for system codes, and the fourth for a mask that determines which processors are to be disabled in a given cycle. The programming environment of the array consists of an assembly language and a library of preprogrammed macros.

Maximum system performance is in the neighborhood of 450 MOPS. Data-transfer rate between the dual-port memory and the array is approximately 340M bytes/sec. A simulation of typical benchmark algorithms indicates that a solution of a linear system of 100 equations requires 22 ms, and a 1024-point, complex (32-bit) FFT is computed in 260 μs. The prototype system is in the final stages of integration and will be tested this summer.



References

1. J.G. Nash, S. Hansen, and W. Przytula, "Systolic Partitioned and Banded Linear Algebraic Computations," *Proc. SPIE* (Real-Time Signal Processing IX), W.J. Miceli, ed., Vol. 698, 1986, pp. 10-16.
2. F. Luk, "A Triangular Processor Array for Computing the Singular Value Decomposition," Cornell Technical Report TR 84-625, July, 1984, Cornell University, Ithaca, N.Y.
3. J.G. Nash, K.W. Przytula, and S. Hansen, "Systolic/Cellular Processor for Linear Algebraic Operations," *1986 IEEE Workshop on VLSI Signal Processing*, S.Y. Kung, R.E. Owen, and J.G. Nash, eds., 1986, IEEE Press, New York, pp. 306-315.
4. M.D. Ercegovac and J.G. Nash, "An Area-Time Efficient VLSI Design of a Radix-4 Multiplier," *Proc. IEEE 1983 Int'l Conf. Computer Design*, Computer Society Press, Silver Spring, Md., pp. 684-687.

P-NAC: A Systolic Array for Comparing Nucleic Acid Sequences

Daniel P. Lopresti
Dept. of Computer Science, Brown University, Providence, RI 02912

The Princeton Nucleic Acid Comparator (P-NAC) is a linear systolic array for comparing DNA sequences. The architecture is a parallel realization of a standard dynamic programming algorithm. Benchmark timings of a VLSI implementation confirm that, for its dedicated application, P-NAC is two orders of magnitude faster than current minicomputers.[1] Experience with the prototype is shaping the design of a second-generation device, to be known as the Brown Nucleic Acid Comparator (B-NAC), that will be algorithmically flexible and more tolerant of fabrication faults.

The primary structure of DNA can be specified as a string of characters chosen from the alphabet $\{A, C, G, T\}$. While there exist a number of different metrics for comparing strings in general and DNA sequences in particular, it is not yet understood which are biologically valid. Nevertheless, an intuitively satisfying measure assumes that DNA "evolves" by undergoing a series of three elemental steps: the deletion of a single "character" (nucleotide), the insertion of a single

$$
\begin{aligned}
&d_{0,0} = 0 \\
&d_{i,0} = d_{i-1,0} + c_{del}(s_i) && 1 \le i \le m \\
&d_{0,j} = d_{0,j-1} + c_{ins}(t_j) && 1 \le j \le n
\end{aligned}
$$

Figure 1. The initial conditions for determining evolutionary distance.

$$
d_{i,j} = \min \begin{cases} d_{i-1,j} + c_{del}(s_i) \\ d_{i,j-1} + c_{ins}(t_j) \\ d_{i-1,j-1} + c_{sub}(s_i, t_j) \end{cases} \qquad 1 \le i \le m, 1 \le j \le n
$$

Figure 2. The recurrence relation for determining evolutionary distance.

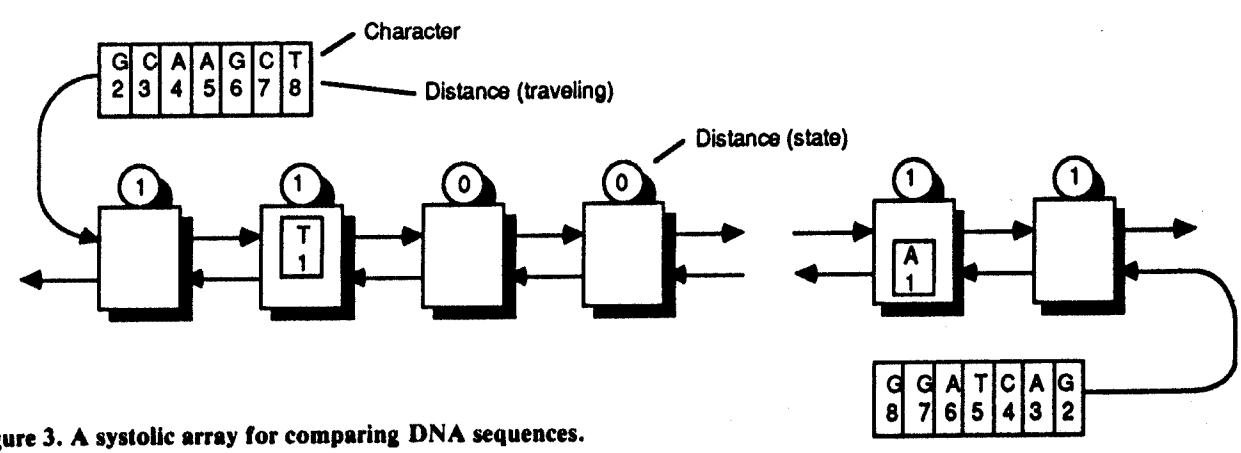


Figure 3. A systolic array for comparing DNA sequences.

character, and the substitution of one character for another. The similarity between two DNA sequences, $S = s_1 s_2 \ldots s_m$ (the *source*) and $T = t_1 t_2 \ldots t_n$ (the *target*), can be quantified by assessing a charge for each of these steps and then finding the least expensive transformation of S into T.

The cost associated with such a transformation is designated the *evolutionary distance*; it can be determined by making use of a well-known dynamic programming approach.[2] Let $d_{i,j}$ be the distance between the subsequences $s_1 s_2 \ldots s_i$ and $t_1 t_2 \ldots t_j$. The initial conditions are shown in Figure 1, and the recurrence relation is shown in Figure 2. The algorithm builds an $m \times n$ table in which the desired result, $d_{m,n}$, appears in the lower right corner.

Fortunately, there is tremendous potential for parallelism in the construction of such a distance table; all values $d_{i,j}$ can be calculated simultaneously for a given $k = (j - i)$. Mapping this recurrence onto a linear systolic array is a straightforward process, although there are several possible dataflows that can yield alternative architectures. One such design is shown in Figure 3. The source and target sequences form two data streams that shift in from the left and right, respectively; one character is input every clock cycle. Initializing distances taken from the first row and column of the dynamic-programming table travel with the characters. As these values pass through the array, they are transformed into distances from the last column and row including $d_{m,n}$. A processor performs one inner-loop step of the recurrence each time it receives a source and target character.

P-NAC is an nMOS implementation of the systolic array just described. To simplify design of the prototype, the general algorithm was restricted by hardwiring the evolutionary costs so that $c_{del}(s) = c_{ins}(t) = 1$ and $c_{sub}(s,t) = 0$ if $s = t$ and $c_{sub}(s,t) = 2$ if $s \neq t$. Each chip contains 30 processing elements, which are snaked in three rows of 10 processors. A bypass option, which shortens the path through a chip to 10 processors, was included so that comparisons of short sequences would not needlessly pass through the entire length of the array. Housed in a 40-pin, 4.6-mm × 6.8-mm standard frame, the device was fabricated using a 4-micron process by the MOS Implementation Service (MOSIS).

By itself, P-NAC is next to useless. A Multibus support board forms the link between the systolic array on the hardware level and the host computer, a Sun 2 workstation, on the software level. A number of benchmarks have been run to test functionality and speed; P-NAC performs 125 times faster than a DEC VAX 11/785 when the VAX uses the same dynamic programming algorithm.[1]

At present, P-NAC's full power is untapped.

Two important issues confronting the designers of VLSI processor arrays are fault tolerance and I/O bandwidth. Because P-NAC is a "conservative" chip (requiring just over 6000 transistors), the former issue is not a major concern; yield has generally been near 100 percent. Nevertheless, P-NAC did suffer a mask defect on its first fabrication run. Fortunately, the bypass option permitted a route around the bad location, which left a usable, albeit a shortened, array. This simple technique for introducing fault tolerance, although not originally intended to serve that purpose, will be improved for use on future versions.

More serious is the problem of I/O bandwidth. While P-NAC is simpler than most parallel architectures proposed in the literature, its appetite for data is insatiable; the chips are 10 times faster than the support board can drive them. For now, this additional power remains untapped.

Despite P-NAC's speed advantage, the prototype implementation suffers because it lacks the flexibility of a software solution; its notion of similarity is preprogrammed and limited. Clearly, evolutionary costs must be permitted to range over some small set of integer values. In addition, it is not necessary for two sequences to be similar in their entirety for their comparison to yield an interesting result. One DNA sequence may resemble a subsequence of another (for example, one gene contained in a larger chromosome), or two DNA strings may share a highly similar subsequence (for example, a common gene). Given two sequences S and T, three questions naturally arise: Are S and T similar? Is there a subsequence S' of S that is similar to T? Is there a subsequence S' of S that is similar to a subsequence T' of T?

It is not surprising that these variations on the basic sequence-comparison problem can be solved by means of dynamic programming algorithms similar to the one just described. A single programmable array could be built to handle any or all of the aforementioned cases. Work is now commencing on such a second-generation device, the B-NAC.

Acknowledgments

Richard J. Lipton, J. Douglas Welsh, and Richard Schaefer, all of Princeton University, played important roles in the development of P-NAC.

References

1. D.P. Lopresti, "Discounts for Dynamic Programming With Applications in VLSI Processor Arrays," PhD dissertation, Computer Science Dept., Jan. 1987, Princeton University, Princeton, N.J., p. 80.
2. R.A. Wagner and M.J. Fischer, "The String-to-String Correction Problem," *J. ACM*, Vol. 21, No. 1, Jan. 1974, pp. 168-173.

Integrating Systolic Arrays Into a Supersystem

Wen-Tai Lin, Chi-Yuan Chin, and Chung-Yih Ho
General Electric Co./CRD, PO Box 8, KWC 524, Schenectady, NY 12301

Recent studies of supercomputers show that the low processor utilization rate displayed by conventional vector machines is mainly the result of traffic between the system memory and processors. Although small-grain architectures (such as systolic arrays and VLSI arrays in general) are designed to ease the memory-processor traffic, they suffer most in that they lack the flexibility to accommodate a wide variety of application algorithms. In this article, we propose a reconfigurable 2D processor array that is a trade-off between the large-grain vector machines and the small-grain, pure systolic arrays. The advantages are fourfold.

- First, when programmable processing elements (PEs) are used for it, many existing systolic algorithms can be directly mapped into the 2D array.
- Second, for large systolic problems and in cases where algorithm decomposition is difficult, it may provide alternative computation modes, such as semisystolic computation.
- Third, when the array is part of the Linc (Link and INterconnection Chip)[1] VLSI interconnection chip, the array can be dynamically reconfigured to accommodate a wide range of computational structures. The mesh-connected Linc array can also be used to simulate many interconnection networks that are used for nonsystolic processing.
- Lastly, the modular 2D array can be degenerated into a 1D array to provide fault tolerance.

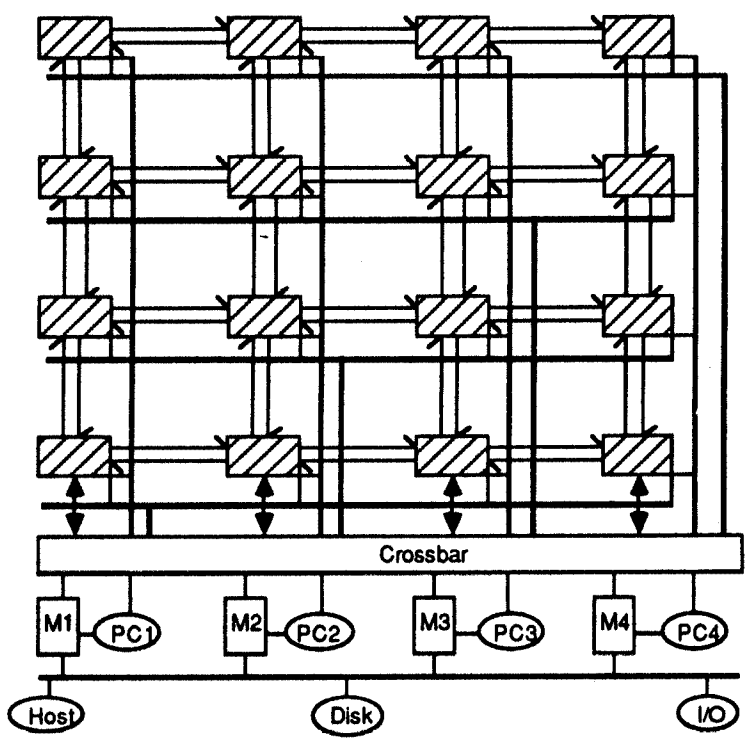


Figure 1. The system architecture of a 4 × 4 Linc array.

The reconfigurable architecture is illustrated by the 4 × 4 Linc array of Figure 1. Each computation node is composed of four Linc chips (which form eight channels, each with 16-bit data paths), two VLSI PEs, two control stores (one for the PEs and the other for Linc chips), and two memory banks (see Figure 2). Each node has four I/O pairs (designated as North, South, East, and West, respectively) connected to its four nearest neighbors and two global buses (the X-Bus and Y-Bus) connected to the processor controllers (PCs). Through the global buses, each PC coordinates either a column or a row of the processor array. To separate the global communication activities from local computational events, the PEs and Linc chips are controlled independently. To configure the PEs with a large degree of freedom, in general, one may use off-the-shelf chips. In a computed tomography image-reconstruction system, each PE has a 32-word × 32-bit register file, a 32-bit multiplier, and a 32-bit ALU. The PEs are microcode-controlled and sequenced by the external PCs. When it operates at a

10-MHz clock rate, the computation node of Figure 2 reaches its peak performance—around 40 MFLOPS.

In the array system, neighborhood communication is provided by the mesh-connected network. When a computation task does not require interprocessor transfer, the Linc resources are allocated so as to fully support the pipelining of Linc's local dataflows. Each Linc node is equipped with an independent set of sequential commands for setting up the Linc data paths. The entire array system can be configured to match a wide variety of computational structures, such as torus, systolic hexagon, systolic ring, binary tree, triangular array, and so on. Irregular computation graphs, such as those used for image rotation and interpolation, can also be easily embedded in the 2D array.[2] To offer PE permutations that are more dynamic, such as those provided in an interconnection network, the array system simulates a variety of interconnection networks by executing the Linc "instruction set." For example, a cube interconnection network for $N = 32$ nodes can be "folded" into a 4×4 Linc array; it takes $2(N/2)^{1/2}-2$ Linc cycles to simulate the five-stage cube interconnection. A single-stage, perfect shuffle-exchange network can be simulated in $\frac{1}{2}(N/2)^{1/2}$ cycles. Since only four I/O pairs are involved, a total of 24 (that is, 4!) Linc control patterns is required.

One often encounters the difficult problem of algorithm decomposition when one is mapping systolic computations into a fixed-processor array. A natural way to overcome this problem, especially in image/signal processing, is to obtain parallelism through the partitioning of input data. For example, a smooth systolic 2D convolution algorithm requires that the convolution kernel size be equal to the number of processors. By partitioning the entire image into subimages and storing each subimage in the local memory of a separate PE, one can derive a semisystolic dataflow that is not hampered by heavy cross-boundary data transfers. In Figure 3, we show that convolution kernel K is broken into four parts—K^i,K^x,K^y, and K^z—when it crosses a subimage boundary. In fact, kernel K can be viewed as wraparound subkernels within the subimage area of each PE. The convolution of each subkernel is carried out separately, with the partial sums denoted as S^i,S^x,S^y, and S^z. After all the subkernels are computed, S^x,S^y, and S^z are forwarded to their respective neighbors to be summed up as a total pixel value.

Since the architecture is reconfigurable, it is a nontrivial task to develop a user-friendly programming environment. However, because of the modular nature of the system architecture and the ease of controlling Linc datapaths, we foresee that software development will be pursued in an empirical manner. The effort will be focused first on the writing of application-specific libraries, then on the development of a high-level language compiler, and finally will be targeted at a fault-tolerant operating system.

References

1. C.-Y. Chin et al., "A Dynamically Reconfigurable Interconnection Chip (Linc)," *Proc. IEEE Int'l Solid-State Circuits Conf.*, Feb. 1987.
2. W.-T. Lin and C.-Y. Chin, "A Reconfigurable Processor Array Based on the Linc Chip," *Proc. Int'l Systolic Array Workshop*, July 1986, Oxford, England., pp. 313-320.

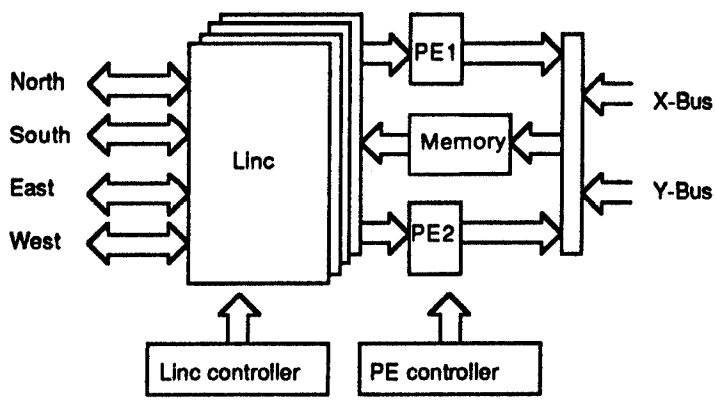


Figure 2. The architecture of a Linc computation node.

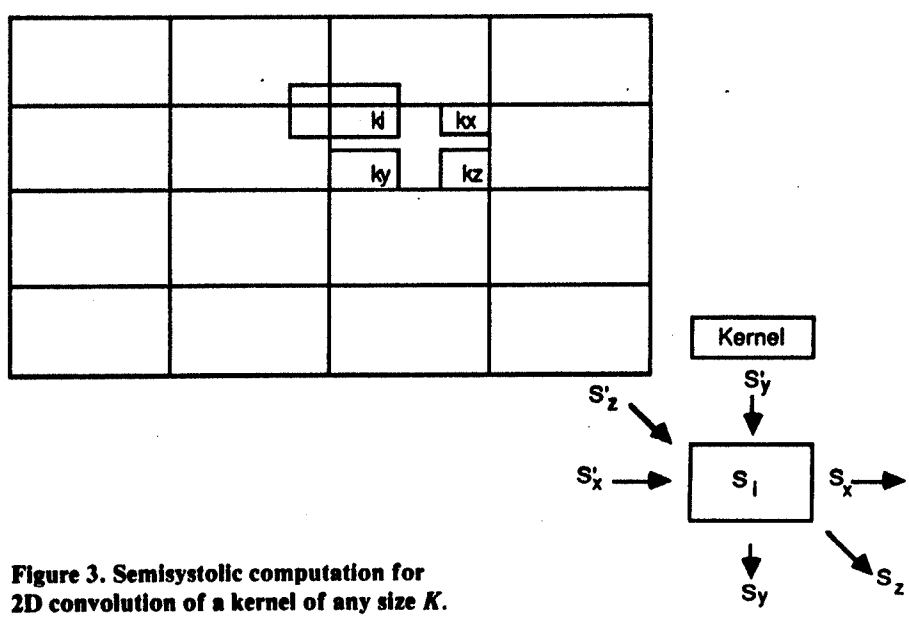


Figure 3. Semisystolic computation for 2D convolution of a kernel of any size K.

The Concept and Implementation of Data-Driven Processor Arrays

Israel Koren
Dept. of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA 01003 (on leave from Technion, Haifa 32000, Israel)

Irit Peled
Dept. of Electrical Engineering, Technion, Haifa 32OOO, Israel

Various topologies and architectural designs for processor arrays have recently been proposed. These designs include systolic arrays and globally asynchronous wavefront arrays.[1] Both array types have a computation front that propagates according to a predetermined control sequence, and consequently these *control-driven arrays* have proven to be very effective for executing highly regular algorithms like vector and matrix operations. There are, however, many computationally demanding problems that do not exhibit high regularity and may therefore prove unsuitable for these control-driven arrays. Still, many of these problems have an inherent parallelism, and it should be possible to exploit this parallelism by means of processor arrays that can provide a high degree of pipelining.

In Koren and Silberman,[2] a new approach to array design was proposed—that of developing specialized array architectures that would be capable of executing any given algorithm. In this approach, the algorithm is first represented in the form of a *dataflow graph* (DFG) and is then mapped onto the array. The processing elements (PEs) in the array execute the operations included in the corresponding nodes (or subsets of nodes) of the DFG; regular interconnections of these PEs serve as edges of the graph.

In general, when an arbitrary algorithm is executed on an array there is no regular propagation of computation fronts. Hence, to speed up the execution of arbitrary algorithms, a more flexible array is needed. Such an array should make possible the generation of new computation fronts and their cancellation at a later time (the time depends on the arriving data operands). We therefore call these arrays *data-driven* arrays. The cell (that is, the PE) in these arrays should be capable of testing for the presence of its operands and executing only the instructions for which all the necessary operands have arrived. Thus, the order in which instructions are executed is data dependent, and the cell is truly a *data-driven* PE.

Processor-array architecture and principles of its operation

The feasibility of designing control-driven arrays was never in question; several types of PEs for these arrays have already been designed (see Fisher et al.[3]). However, the degree of hardware complexity required to add the data-driven property was not clear to us. Therefore, we made a preliminary design of an appropriate processing element. The result of this design is very encouraging: The total hardware complexity of the cell—which is presented next—is less than 9000 transistors in NMOS technology. This low complexity should make possible the fabrication of a VLSI chip containing about 50 to 100 cells. The first phase of the design has already been completed and is reported in Peled.[4] A group of graduate students in the Dept. of Electrical and Computer Engineering at the University of Massachusetts in Amherst is now finalizing the detailed design and layout of the VLSI chip.

The proposed floorplan of the chip is shown in Figure 1. The floorplan contains

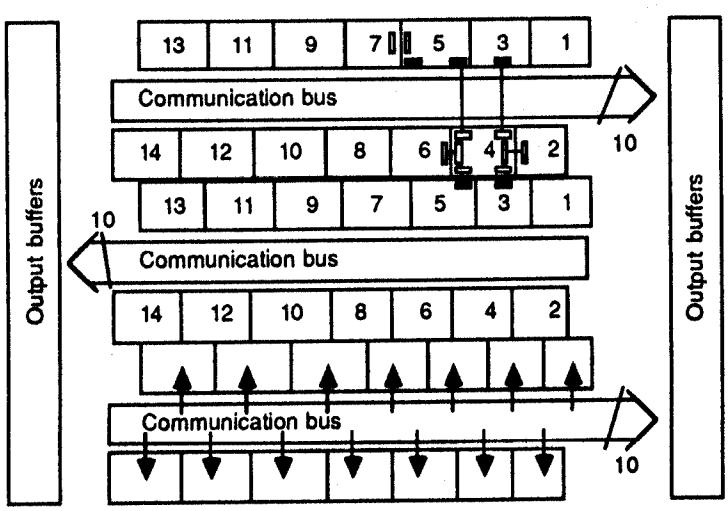


Figure 1. The floorplan of the processor-array chip.

data-driven cells arranged in rows in such a way that the typical cell has six immediate neighbors in a hexagonally connected processor array. The cells communicate with an external host computer through buses, as shown in Figure 1. In this host, the programs of the individual cells are prepared and are then distributed to the cells. The host also supplies the input operands and accumulates the final results. Unlike host-to-array communication in control-driven arrays, restricting the host-to-array communication to passing only through boundary cells can slow down the array's operation substantially. Consequently, we have decided to allow each cell to communicate directly with the host.

The functional blocks that make up the basic cell are depicted in Figure 2. There are six registers, R_1 through R_6, which are connected to a common internal bus; each of them is also directly connected to its corresponding register in one of the six neighboring cells (see Figure 2). The latter connection is under hardware control, since its timing is crucial. In our design, this data transfer takes a single clock and is done *in parallel* with all other operations in the cell.

The *instruction memory* contains six instructions that specify the cell's operations; one instruction per register (out of R_1 through R_6 registers). Having six instructions per cell increases the level of utilization of the cell and leads to a lower overall execution time.

The *flag array* is a uniquely designed block that makes possible the data-driven operation of the cell. The instructions in the cell are not executed in any predetermined order. Instead, the arrival of all operands for a certain instruction enables the cell to execute set instructions. The flags monitor the movement of operands both within the array and in and out of the cell. For each register there is a flag indicating whether the register has an operand or whether it is empty and can receive a new operand. Only a single cycle is needed to test these flags to determine whether an instruction is ready to be executed.

The other functional units in the cell are self-explanatory.

In parallel with designing the array, a procedure for mapping DFGs onto data-driven arrays has been developed and programmed by Mendelson and Silberman.[5] In this procedure, the user's program (in VAL) is translated into a DFG and then mapped onto a finite array of PEs. The procedure allows us to take advantage of the data-driven cell's capability of performing up to six operations. A node in the original DFG includes only a single operation. Therefore, we may combine up to six simple neighboring nodes and map them onto a single PE. Finally, the array is partitioned into several subarray chips according to the technology-imposed limitations on the number of PEs per chip.

In summary, the idea of directly mapping an arbitrary algorithm onto a VLSI array has been shown to be feasible. Further research is now being carried out to prove the effectiveness and practicality of this approach.

References

1. S.Y. Kung, "On Supercomputing With Systolic/Wavefront Array Processors," *Proc. IEEE*, July 1984, pp. 867-884.
2. I. Koren and G.M. Silberman, "A Direct Mapping of Algorithms Onto VLSI Processor Arrays Based on the Data Flow Approach," *Proc. 1983 Int'l Conf. Parallel Processing*, Aug. 1983, pp. 335-337.
3. A.L. Fisher et al., "Design of the PSC: A Programmable Systolic Chip," *Proc. Third Caltech Conf. VLSI*, March 1983, California Institute of Technology, Pasadena, Calif., pp. 287-302.
4. I. Peled, "A Data-Driven Processing Element," master's thesis (in Hebrew), 1986, Dept. of Electrical Engineering, Technion, Haifa, Israel.
5. B. Mendelson and G.M. Silberman, "Mapping Data Flow Programs on a VLSI Array of Processors," *Proc. 1987 Int'l Conf. Computer Architecture*, June 1987. Also published as a technical report in 1987 by the Dept. of Computer Science, Technion, Haifa, Israel.

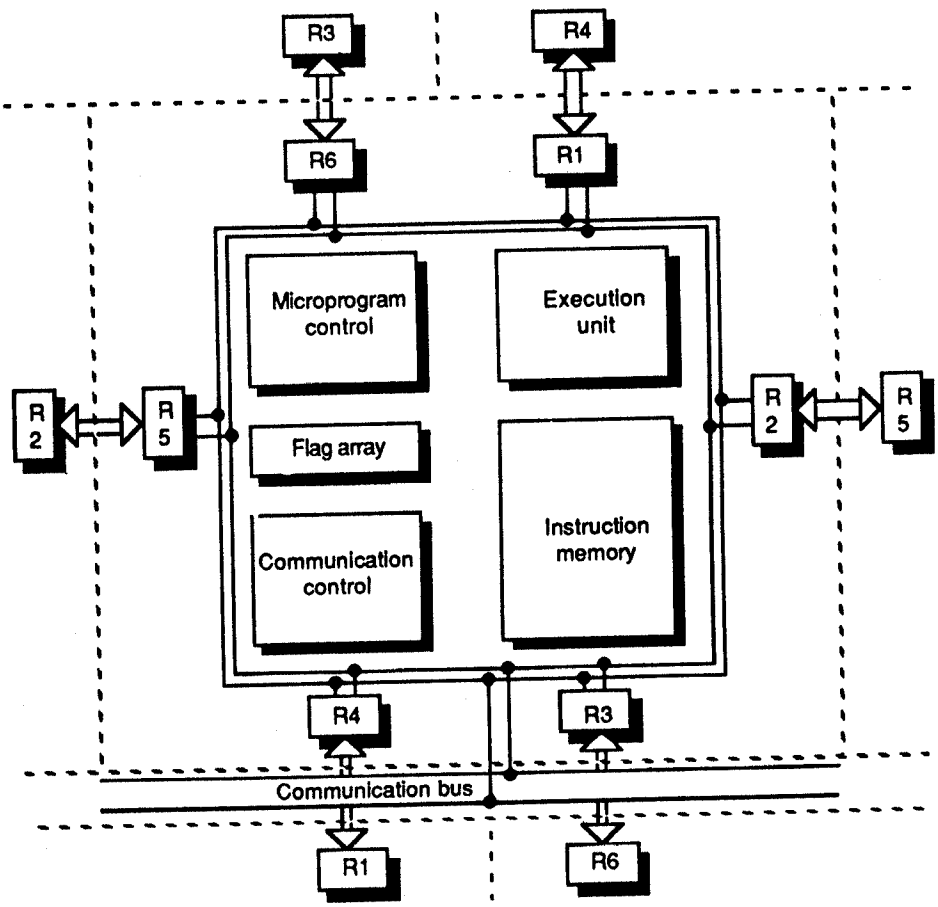


Figure 2. Functional blocks in the data-driven cell.