QUENTIN E. DOLECEK

QUEN: THE APL WAVEFRONT ARRAY PROCESSOR

Developments in computer networks are making parallel processing machines accessible to an increasing number of scientists and engineers. Several vector and array processors are already commercially available, as are costly systolic, wavefront, and massive parallel processors. This article discusses the Applied Physics Laboratory's entry: a low-cost, memory-linked wavefront array processor that can be used as a peripheral on existing computers. Available today as the family of QUEN processors, it is the first commercial parallel processor to bring Cray 1 computation speeds into the minicomputer price range.

INTRODUCTION

The QUEN is an implementation of the memory-linked wavefront array processor[1] (MWAP) technology developed at the Applied Physics Laboratory with independent research and development funding. Based on the concept of waves of computation traveling through an array of processors,[2] it was created to provide high-speed solutions of numerically intensive computational algorithms. In its most general form, shown in Figure 1, the array is configured as an N-dimensional mesh of processors, each operating as an independent unit that executes instructions stored in its private, local program memory. Data for each processor are contained in multiport memories connected to the adjacent processor on its boundaries. Computation and data flow in the mesh are controlled with hardware synchronization structures (flags) in each multiport memory.

Because each processing element in the array has a large memory in its data path, a simple linear array of processors can implement a wide range of data topologies. For example, the linear array can operate as a two-dimensional array by implementing a column of the array at each node, as shown in Figure 1, or it can operate as a three-dimensional array by implementing a vertical plane of the array at each node. Thus, linear MWAP arrays can be used for the high-speed computation of many problems.

QUEN is the trademark of Interstate Electronics Corp., Anaheim, Calif., for a family of MWAP's being marketed for military applications and for commercial use as attachments to VAX/VMS host computers and SUN work stations. The members of this family of processors are differentiated by the number of processing elements in a system. The largest unit is a QUEN 64, providing 1.28 billion floating-point operations per second. Two smaller QUEN 8 units, each providing 128 million floating-point operations per second, are installed at APL; one, in the Kossiakoff Center, is installed on the JHU/APL computer network and is open for general use, and the other is installed in the Sonar Program Analyzer (SPAN) laboratory for use in sonar signal and image processing.

QUEN MWAP ARCHITECTURE

The QUEN MWAP is a high-speed programmable processor consisting of a host computer interface and a linear array of processing elements interleaved with dual-port memories (DPM'S). Figure 2 shows a system block diagram of the QUEN MWAP. It uses a multiple-instruction multiple-data architecture at its array architecture level, allowing both medium- and coarse-gain parallelism to be used at the array level. Each node in the processor consists of a DPM and a horizontally microprogrammed single-instruction, multiple-data processing element, which enables parallelism in computations to be used at the processing element.

Each QUEN processing element provides fixed- and floating-point operations on both 32- and 64-bit data, with hardware-supported multiplication, addition, subtraction, and logic operations. The processor element also gives hardware support for floating-point divide and square-root operations. The element uses 64-bit-wide instruction

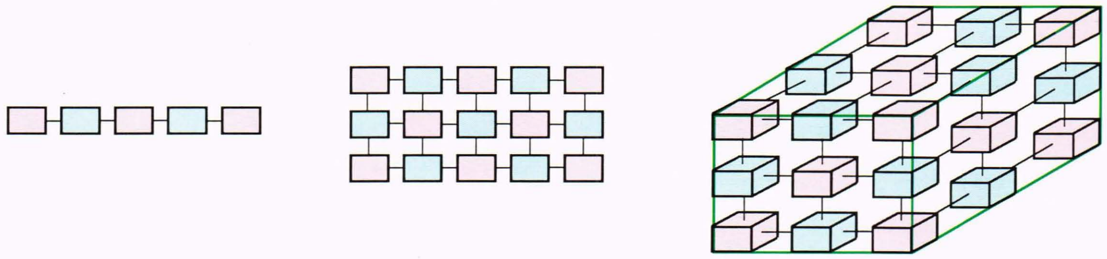


Figure 1. One-, two-, and three-dimensional MWAP configurations, showing dual-port memory (red) and processor elements (blue).

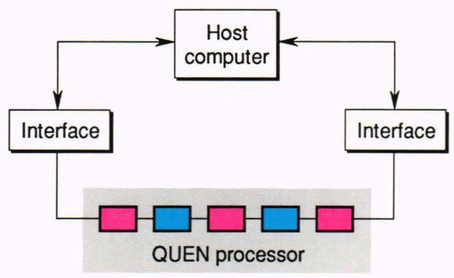


Figure 2. Basic QUEN system architecture, with dual-port memory (red) and processor elements (blue).

words with five separate operation-code fields. This permits simultaneous execution of a left and right DPM operation; a multiplier operation; an arithmetic unit operation such as addition or subtraction; and a conditional branch operation based on a loop counter or arithmetic test result, with loop counter and data address modifications. Instructions executed by the processing element are fetched from a local program memory over a separate 64-bit bus. The fetch and execution of instructions are overlapped for efficiency, with zero instruction delay between the detection of a branch and the execution of the instruction at the branch address. This highly parallel structure in the processing element allows the overhead of array indexing, loop counting, and data input and output to be performed simultaneously with up to two arithmetic operations; in fact, the processing element can execute loops that consist of one instruction. (The instruction jumps back to itself until the loop counter expires.) This parallelism results in much faster execution of programs than with conventional architectures. Each processing element can sustain computation rates approaching 16 million floating-point operations per second.

The DPM's provide local data storage, synchronization, and interprocessor communications. The processors connected in each side of the memory can access (read or write) to the memory simultaneously. The two memory ports, in a single DPM, operate asynchronously, allowing each processor to run independently and permitting the array to be extended to any length. A new memory operation can be initiated at each port of every processor instruction cycle, matching the memory bandwidth to the processor bandwidth. In addition, each memory port performs address calculations, using an address generator as shown in Figure 3. Each address generator contains 16 independent register pairs and an address modification circuit. In each pair, the base register is loaded once, and the address register is modified during each memory access by the value specified in the base register. This modification can increment, decrement, bit-reversed-address increment, or reset the address

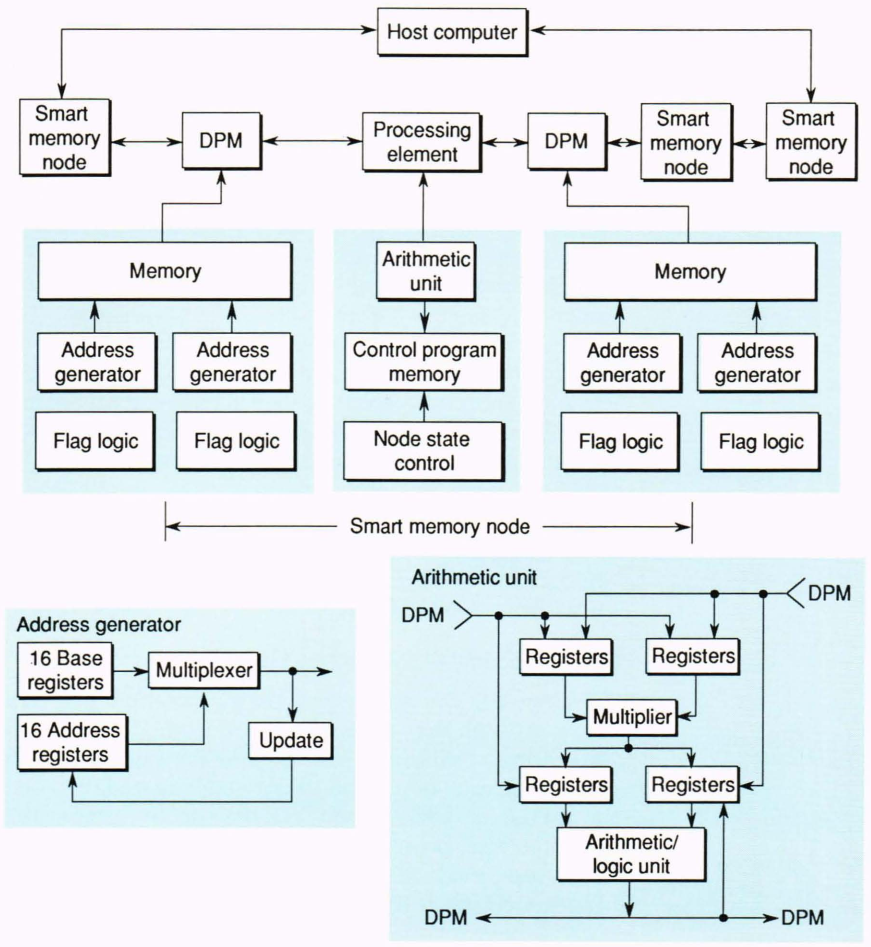


Figure 3. QUEN node structure.

register to the base address. All memory operation addresses are handled by this structure. Finally, each DPM contains two flags by means of which each processing element can control access to blocks of shared memory through instructions that allow it to set or reset the flags and hold instruction execution on the basis of flag tests.

The interface of the MWAP to the host computer provides bidirectional communication and control of the array through the first and last DPM's. Programs are loaded into the array by placing the program code in the first DPM and propagating the code to the appropriate node. Computation is done by placing data in the first (and possibly last) DPM and placing the array in the run mode. These load and compute functions are controlled by activating the array processing states described in the boxed insert. In addition, the host controls computations by access to the first and last DPM flags.

The foregoing description of hardware interconnections and operation of the MWAP architecture obscures the central MWAP concept of a computation wavefront traveling down the array. Each node in the array modifies a term in the wavefront, or transforms the wavefront information, and acts as a secondary source responsible for propagation of the wavefront. Computation wavefronts can be used to compute individual terms in a function, recursions in an algorithm, or sequential algorithms on the data in the wavefront. The concept is similar to a wavefront traveling through water or air; as the wave moves through the medium, it is modified by the medium. For the MWAP, the medium is the memory containing the data. Thus, the MWAP processor can be conceptualized as "smart" memory propagating a computation from its input to its output, as shown in Figure 4.

MWAP PROGRAMMING

To obtain the optimum performance from any computer, the program must always be designed to suit the architecture of the computer, even on serial computers such as the VAX machines, vector computers such as the Cray 1, and parallel machines such as the NCube. That is why a carefully written assembler code, which takes into account the structure of the computer, can still outperform the code produced by the most sophisticated compilers. What has changed with the advent of parallel computers is the ratio between the performance of a good and a bad computer program. The ratio is not likely to exceed a factor of two or three on a serial machine, whereas factors of ten and more are common on parallel computers.[3]

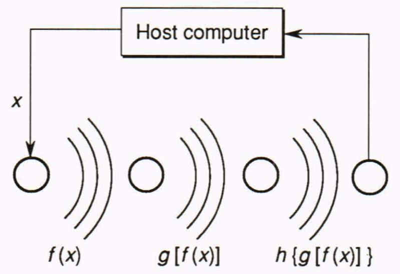


Figure 4. Wavefront computation concept used in the QUEN. Computation waves are modified at each memory node as they propagate through the processor (circles represent smart memory nodes).

QUEN PROCESSOR STATE DESCRIPTIONS

Reset

Causes all processing elements and DPM to go into the reset state; that is, address register zero for each DPM is set to zero and put in the increment by one normal addressing mode, while flag one in each left DPM is reset (enabling each processing element to access the memory to its right).

Pause

Causes each processing element to go into the pause state; that is, each element stops normal execution.

Load

Causes each processing element to go into the load state, wherein each element attempts to load the information from the left DPM, using address register zero, into the element's control random access memory. This process will not start until flag one in the left DPM is set. Also, if the element is in the load state and the host issues a run command, the element will not go into the run state until the load has been completed.

Run

Causes each processing element to go into the run state; that is, each element starts normal execution.

We have implemented application programs on the QUEN MWAP in many areas, including image processing, signal processing, scientific computation, and artificial intelligence. Each program was written in the C language or Fortran language using subroutine calls for QUEN computations. The QUEN subroutines were written in the QUEN language, which resembles the C language. The QUEN assembler and linker were then used to obtain files for loading into the QUEN processor. A subroutine call from the host language then loaded the program file into the QUEN, and computations were done by QUEN calls to send and receive data from the processor. For all application programs, the performance improvement over using a VAX 11/780-class computer alone was not less than a factor of 50. Speed improvement factors of several hundred were obtained for problems implemented with wavefront propagation.

PROGRAMMING METHODS

There are two basic methods for programming data flow through the MWAP: the block method, in which a problem is partitioned into computational blocks processing at each node, and the cascade method, in which a stream of data is sent to the MWAP, propagated through each processor node, and returned to the host. Any computation can be done on the MWAP using the block method. If the computation time in each processing element is equal to or significantly greater than the data input/output time, the method will be effective and will improve the performance of the host machine. The cascade method achieves the highest use of MWAP hardware resources and the highest computation speed, since data are being passed and processed concurrently. It includes programming the MWAP as a systolic array, a data-flow array, and a wavefront array. It is more difficult than the block method because a cascade-type algorithm must be found for the computation. This type of algorithm does not exist for all computations and is not usually obvious when it does exist. Thus, ultrahigh-speed computation has a price.

As an example of the block method of MWAP programming, consider the parallel canonic-form digital filter shown in Figure 5. The data are passed (pipelined) down the array to all except the last node. Computations begin in each node once all the nodes have received the correct number of data points. After all computations are complete, the results are passed to the last node, which sums the results and sends the completed output to the host.

Contrast this with the cascade method, shown in Figure 6. Here, the MWAP receives a stream of data, which goes to the first node in the array. This node transforms the data and passes the result to the next node. Each node in turn receives the output of the previous node, transforms the data, and sends the result to the next node. The final result exits the last node to the host computer. During node computations, the data are simultaneously transformed and passed. We have improved performance by matching the topology of the algorithm to that of the MWAP, with each node implementing a stage in the digital filter.

The cascade method can also be used to implement systolic algorithms on the MWAP. Nodes are assigned and synchronized to pump computations rhythmically through the array in equal time slots. For a digital filter, this can be done by expressing the filter as a difference equation:

$$Y_z = \sum_{k=0}^{N} B_k Z^{-k} X_z - \sum_{m=1}^{M} A_m Z^{-m} Y_z \ .$$

Writing the first four time terms for $N = M = 3$,

$$\begin{aligned}
Y_0 &= B_0 X_0 \\
Y_1 &= B_0 X_1 + B_1 X_0 \qquad - (A_1 Y_0) \\
Y_2 &= B_0 X_2 + B_1 X_1 + B_2 X_0 - (A_1 Y_1 + A_2 Y_0) \\
& \qquad\qquad\qquad X \text{ wave} \qquad\qquad Y \text{ wave} \\
Y_3 &= B_0 X_3 + B_1 X_2 + B_2 X_1 - (A_1 Y_2 + A_2 Y_1)
\end{aligned}$$

$$BX \text{ wave} \rightarrow \qquad \leftarrow AY \text{ wave}$$

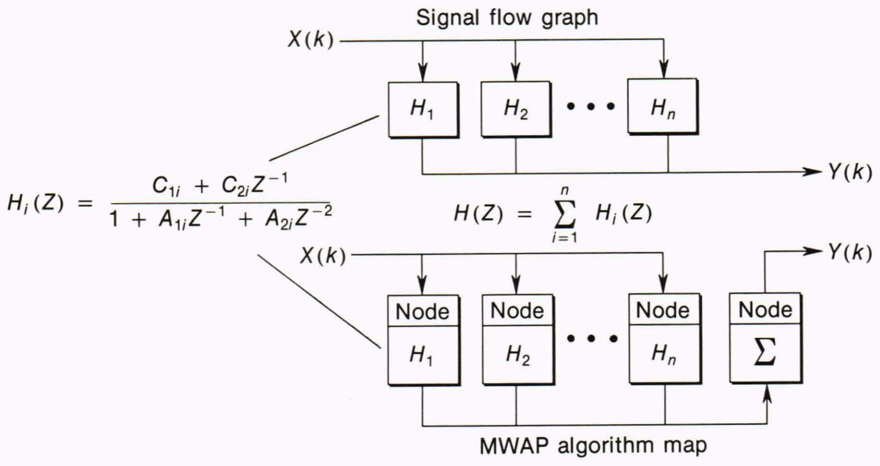

Signal flow graph
X(k)
H1
H2
Hn
Y(k)
$H_i(Z) = \dfrac{C_{1i} + C_{2i}Z^{-1}}{1 + A_{1i}Z^{-1} + A_{2i}Z^{-2}}$
$H(Z) = \sum_{i=1}^{n} H_i(Z)$
X(k)
Node
H1
Node
H2
Node
Hn
Node
Σ
Y(k)
MWAP algorithm map

Figure 5. Parallel canonic filter implementation on the QUEN. (Z^{-j} implies X is delayed j times; A and C are filter coefficients.)

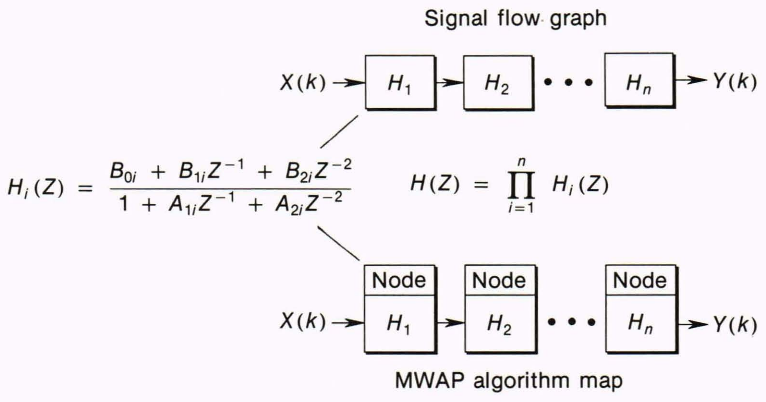

Signal flow graph
X(k)
H1
H2
Hn
Y(k)
$H_i(Z) = \dfrac{B_{0i} + B_{1i}Z^{-1} + B_{2i}Z^{-2}}{1 + A_{1i}Z^{-1} + A_{2i}Z^{-2}}$
$H(Z) = \prod_{i=1}^{n} H_i(Z)$
Node
Node
Node
X(k)
H1
H2
Hn
Y(k)
MWAP algorithm map

Figure 6. Cascade canonic filter mapping to the QUEN. (Z^{-j} implies X is delayed j times; A and B are filter coefficients.)

We see that the systolic digital filter can be implemented using four propagating wavefronts: a wavefront of inputs X propagating to the right, a wavefront of inputs Y propagating to the right, summations of BX propagating to the right, and summations of AY propagating to the left.

This systolic computation can be done on the MWAP, since data can flow in either direction in the array and timing can be imposed using the node flags. The resulting computation flow is shown in Figure 7. Time in the figure is the number of instructions it takes to read two data items, write two data items, and compute the $AX + B$ summation. The entire set of operations requires two instructions in the MWAP. Noting that every other time cycle must be idle to permit computation and feedback of the BY summation terms in synchronization with the AX summation terms, the systolic implementation computes a result every four instruction cycles, or 2 million results per second.

The cascade method can also be used to implement wavefront algorithms. Here the nodes in the array require only assignment of computing tasks, since computation at each node takes place only when all required data for the node are available. Implementation of the digital filter with a wavefront algorithm is shown in Figure 8. The same computations and wavefronts are used as in the systolic algorithm, but idle time for synchronization is not required. Thus, the wavefront method is twice as fast as the systolic method.

FAST FOURIER TRANSFORM

Much of the current revolution in the application of digital signal processing is a result of the ability to map signals into the frequency domain efficiently. The basic equation for the discrete Fourier transform (DFT) is defined as

$$X(m) = \sum_{k=0}^{N-1} x(k) W_N^{mk} , \qquad (1)$$

where $m = 0,1, \ldots ,N - 1$, and $W_N = \exp(-j \cdot 2\pi/N)$.

Direct computation of this transformation, for $N = 4196$, requires about 10 min on a machine such as the IBM 7094; by contrast, the same transformation using a class of algorithms known as the fast Fourier transform (FFT) requires about 2.5 s, and one using the MWAP requires less than 0.75 ms.

The key to FFT's is to reduce or eliminate the redundancy in the DFT equation. The cyclic nature of W_N^{mk} creates this redundancy (see Fig. 9) and is reduced by dividing the sequence into smaller ones. One strategy, the decimation-in-time approach, divides the sequence into odd and even sample sequences. An N-point sequence can then be transformed by combining the DFT's of these two $N/2$ sequences:

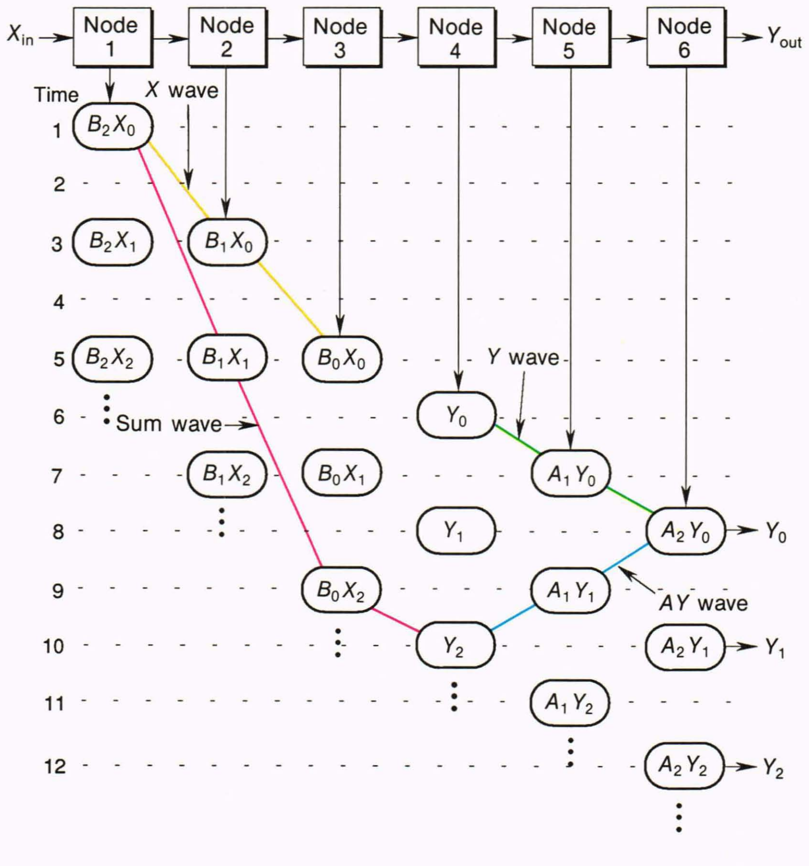


Figure 7. Systolic computation of a digital filter.

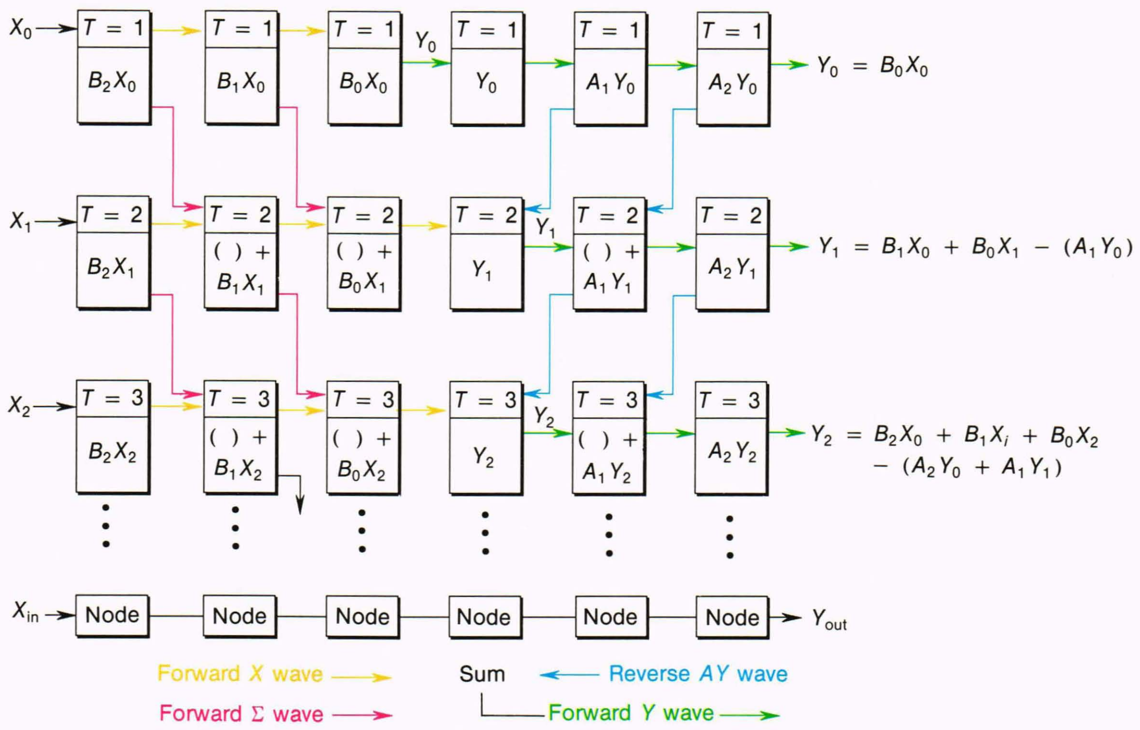
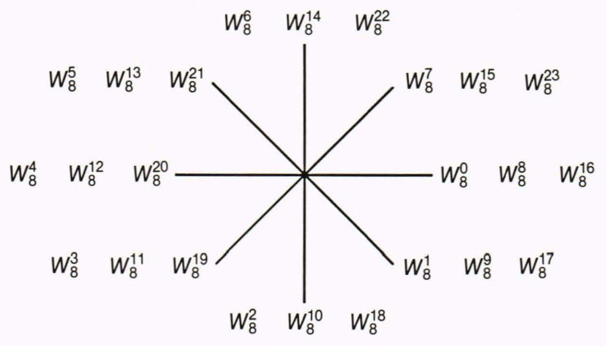


Figure 8. Wavefront propagation for a digital filter.



W_8^6 W_8^{14} W_8^{22}
W_8^5 W_8^{13} W_8^{21}
W_8^7 W_8^{15} W_8^{23}
W_8^4 W_8^{12} W_8^{20}
W_8^0 W_8^8 W_8^{16}
W_8^3 W_8^{11} W_8^{19}
W_8^1 W_8^9 W_8^{17}
W_8^2 W_8^{10} W_8^{18}

Figure 9. Cyclic nature of the exponential function W_N^{mk} (see Equation 1) in the Fourier transform.

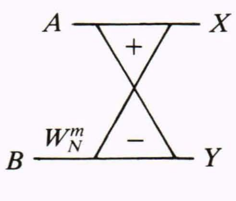
$$x_1(k) = x(2k)$$
$$k = 0, 1, \ldots, (N/2) - 1$$
$$x_2(k) = x(2k + 1) \ ,$$

yielding $X(m) = X_1(m) + W_N^{mk} X_2(m)$.

The division into odd and even sequences, shown in Figure 10, is continued until the problem is reduced to computing and combining a series of two-point transforms called butterflies:

$$X = A + W_N^m B$$

$$Y = A - W_N^m B$$



To compute the FFT, the butterfly computation is done $N/2$ times in each MWAP node as it computes one column, or stage, of the FFT. As each node completes its computation of a stage, the result is propagated to the next node. Thus, for each Fourier transform, a wave of computations flows down the array, starting with the left-most node in the array and ending in the right-most node. Because each node operates on a transform wavefront once and is then free to accept a new wavefront, multiple Fourier transform waves can be traveling through the array simultaneously. Thus, a new FFT can begin every $N/2$ butterfly-computation times when log N nodes are used to compute the required $(N/2)$ log N butterflies per FFT.

The FFT illustrates a wavefront with changing shape as it travels down the array. Each node modifies, or transforms, the data in the wavefront and modifies the order of the data, or the shape of the wavefront. The changing of wavefront shape is handled by the address generators in each node. Read and write sequences are modified at each memory boundary. In Figure 10, each node reads on the left from $A(0)$, $B(0)$ and then from $A(1)$, $B(1)$, etc., but writes to the right in $A(0)$, $A(1)$ and then in $B(0)$, $B(1)$, etc. Four address generator registers are used at each node boundary to control the two complex data buffers, A and B. This implements the correct data sequences for the computation.

The FFT also illustrates an important feature of MWAP architecture—the ability to perform bit-reversed addressing. The algorithm requires that either the input or the output sequences be reordered as shown in Figure 11. The MWAP implements this reordering with reverse-carry addition in the address generator. Here the address

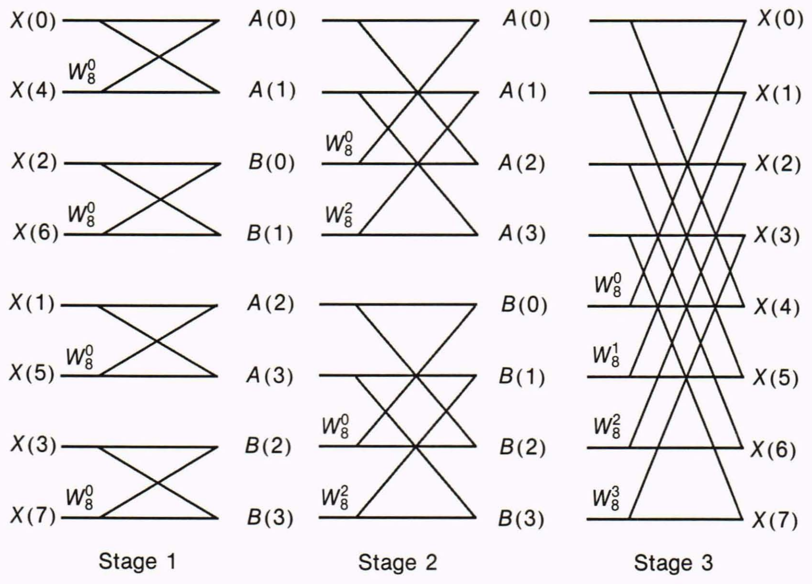


Figure 10. Decimation-in-time FFT (N = 8).

	Binary address	Bit-reversed address		Reverse-carry addition
$X(0)$	000	000	$\rightarrow X(0) \leftarrow$	000
				+1
$X(1)$	001	100	$\rightarrow X(4) \leftarrow$	100
				+1
$X(2)$	010	010	$\rightarrow X(2) \leftarrow$	010
				+1
$X(3)$	011	110	$\rightarrow X(6) \leftarrow$	110
				+1
$X(4)$	100	001	$\rightarrow X(1) \leftarrow$	001
				+1
$X(5)$	101	101	$\rightarrow X(5) \leftarrow$	101
				+1
$X(6)$	110	011	$\rightarrow X(3) \leftarrow$	011
				+1
$X(7)$	111	111	$\rightarrow X(7) \leftarrow$	111
				+1
				000

Figure 11. Generation of bit-reversed addresses using reverse-carry addition.

generator modifies an address by adding the increment value to the memory address, and the carry bit is propagated from the most significant to the least significant bit. The result is cyclic bit-reversed addresses as shown in the figure. This addressing node is required for FFT and other transforms. It can also be used to generate cyclic addresses for implementing sorting, stack, and feedback loop operations.

FRACTALS

Consider the iterative equation $Z = Z^2$ in the complex number plane. If Z begins as a number inside the unit circle, it will iterate toward 0; any number chosen outside the circle wlll iterate toward infinity; and any number chosen on the circle will iterate to some other number on the circle. Every equation of the form $Z = f(Z,C)$, where Z and C are complex numbers, poses two questions. The first is: for all possible starting Zs and a constant starting C, what happens when Z is iterated? This is the Julia plane, named after Gaston Julia, a French mathematician. The second question is: for all possible Cs and a constant starting Z, what happens when Z is iterated? This is the Mandelbrot plane, named after Beniot Mandelbrot, a mathematician at IBM. Fractals are a pictorial representation of either one of these questions.

Mandelbrot fractals were obtained on the QUEN by computing the number of iterations required for the complex function to go to infinity:

$$Z = Z^2 + C ,$$

where $C = X + iY$, and X and Y are the coordinates of an image pixel.

We of course did not wait for the numbers actually to go to infinity, but to some escape value T. If Z escaped within some number of iterations N, the X,Y pixel was assigned a color representing the number of iterations required for escape. If not, it was assigned the color black.

The algorithm is deceptively simple. Only a few inputs are required: the origin coordinates for the image, the increments for X and Y, the number of pixels to compute in X and Y, and the maximum number of iterations, N. But the problem requires massive computation and results in large output number sets. For example, a typical 400 × 400 pixel image, with N set at 1000 or more, results in 160,000 output data points and requires Z to be computed and tested hundreds of millions of times. This can be seen in the Fortran-like code fragment below:

```
FOR 30 I = 1,400
Y = Y + Yinc
  FOR 30 J = 1,400
  Z = complex (0,0)
```

```
          X = X + Xinc
          C = complex (X,Y)
            FOR 10 k = 1, N
            Z = Z*Z + C
            If ((REAL (Z)**2 + IMAG(Z)**2).GT.T)
              GOTO 20
10          continue
          Set pixel color black
          Go to 30
20        Set pixel color for value k
30      continue
```

The problem is not only computation bound, it also cannot be formed as a parallel algorithm using many processors to compute and test the function Z. The image can, however, be generated on the MWAP using the following block programming method:

1. The host sends origin coordinates, increments, and N to the first node.
2. Each node passes this information to the next node, with the X origin offset by 50 increments.
3. The first node computes 50 points in X, including color transformation, and writes the results to its right. It then sets a ready flag to the right. Each successive node repeats this process and copies the results of the previous node to the right when it receives a ready flag from the left. It then waits for a clear flag from the right.
4. The last node releases a 400-pixel X scan to the host and sends a clear flag to the left.
5. Each node increments Y, and steps 3 and 4 are repeated until the image is complete.

This procedure computes one-eighth of the image in each MWAP node and requires 21 MWAP instructions. The code segment below illustrates how the host uses MWAP from the Fortran language:

```
        Call QUENINIT (MANDEL)
          Read X, Y, IncX, IncY, N
          Call QWRITE (X, Y, IncX, IncY, N)
            FOR 20 I = 1, 400
            Call QREAD (xdat)
              FOR 10 J = 1, 400
10            Plotdata(I,J) = xdat (J)
20        Continue
        Call Plot (Plotdata)
```

The fractal shown in Figure 12 was done on a MicroVAX II work station, with N set to 10,000. It required over 4 h to compute using the MicroVAX II alone. Using the MWAP, the image was generated in 2.5 min.

FEATURE EXTRACTION FROM IMAGES

In some fields, we still cannot write mathematical equations that accurately describe processes of interest. If you were asked to describe the picture shown in Figure 13, you would probably say that it looked like a group of lines with a herringbone pattern in the background. This observation is easy for you, yet almost impossible for a modern digital computer. Part of the problem is that we do not yet fully understand the algorithms of thinking. But part of the problem is also computation speed. The APL Strategic Systems Department is applying the MWAP to image understanding using three techniques: pattern matching, cellular logic operations, and neural nets.

In pattern matching, the inner product of pattern template weights W and image pixel vectors X are comput-

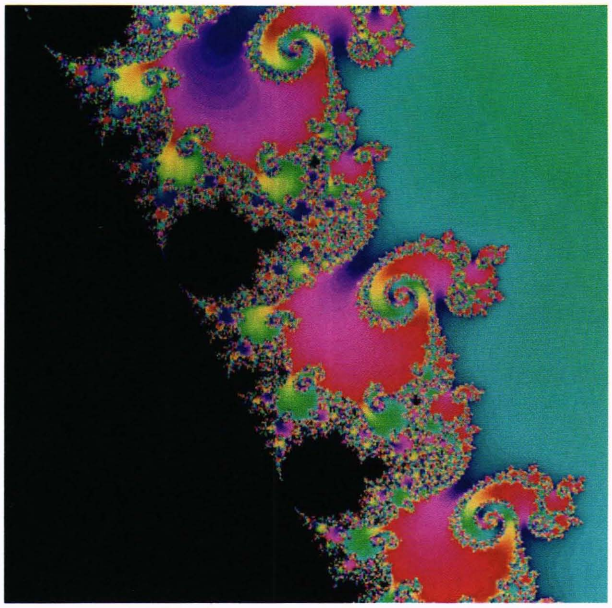


Figure 12. Mandelbrot fractal computed on the QUEN processor and displayed on a MicroVAX II work station.

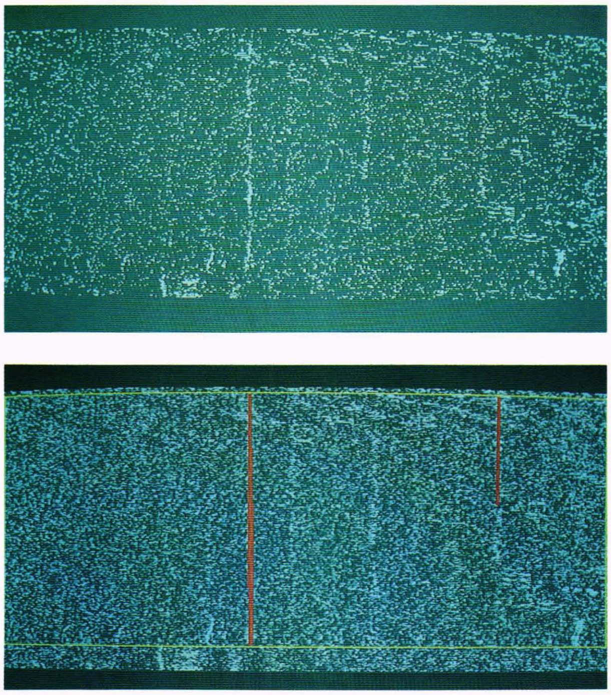


Figure 13. Feature extraction from gray-scale images using the QUEN (top, unprocessed; bottom, processed).

ed, and a pattern is detected when a specified threshold is exceeded: $W \cdot X > T$ pattern is present.

For example, suppose the center of the point template, shown in Figure 14, is moved around the image from pixel to pixel. At every position, we multiply every point of the image inside the template by the number indicated in the corresponding entry of the template and sum the results. If all image points inside the template area have the same value, the sum is zero. If not, the sum is different from zero and maximizes when the template is centered on a point feature. Thus, point features can be found by thresholding the inner product of the image and template as the template is swept through the picture. This concept can be extended, by using different templates, to detect various image features and can even detect transitions in the image by computing the two-dimensional gradient of the image. The MWAP has been programmed to perform template matching on images with up to 1024×1024 pixels using template sizes from 3×3 to 16×16.

Noise in images can be removed by cellular logic filters and by the augment and reduction operators. In a reduction operation, objects (pixels) are replaced by background elements if none of their immediate neighbors are objects. Conversely, augmentation causes background elements to be replaced by object elements if there are object elements in their neighborhood. A sequence of some number of reductions Q, followed by Q augmentations, removes "noise objects" of maximum dimension $2Q$. The reverse procedure fills in object regions $2Q$ in size. These two procedures have been used on the image in Figure 13 to delete localized noise and connect vertical line segments. The result, shown in the figure, was computed and colored by the MWAP in less than 1 s.

The problem in extracting information from sonar images has been precisely stated by Michael Roth.

> Pattern recognition of fixed patterns in stationary backgrounds is a straightforward task for which numerous effective techniques have been developed. If the patterns or the backgrounds are variable in either a limited or known manner, more complex techniques—such as those using methods based on artificial intelligence—can be effective. But if the patterns or backgrounds vary in an unlimited or unknown manner, the traditional approaches have not been able to furnish solutions.[4]

Expert systems, to date, have been unable to solve the feature extraction problem for sonar images, because analysts have not been able to define an effective rule set.

A layered neural-network approach is being developed to either solve the problem or yield effective rule information for an expert system. The network, shown in Figure 14, uses feedforward connections to analyze images and feedback connections to learn image characteristics and features. The concept is simple: first "show" the network a series of sonar images and their correct analysis, then determine how well the network learned the solutions by showing it new images. Sonar image analysis requires the synthesis of information across many images, each looking in a different direction for the same segment of time. A single "snapshot" in the sonar environment may consist of 50 images, each containing 1 million pixels of multibit information. Thus, we have a computation-bound problem of first magnitude, and development to date has been limited by the number and size of images that can be processed in a realistic amount of time.

We are currently programming the QUEN to perform as a feedforward, feedback neural network in which each layer is represented by a memory node in the array. Each layer of the network will be able to contain up to 10,000 neural nodes, have full bidirectional links with the adjacent layers, and handle both binary and multilevel data representations. The QUEN 8 will handle up to 8 layers; the QUEN 16, when delivered to APL, will handle up to 16 layers. We estimate that either QUEN will perform 1 million iterations of the network per second.

SUMMARY

The QUEN is a high-speed, multiple-instruction, multiple-data MWAP for use as a peripheral on a host computer or as a computation unit in a data processing

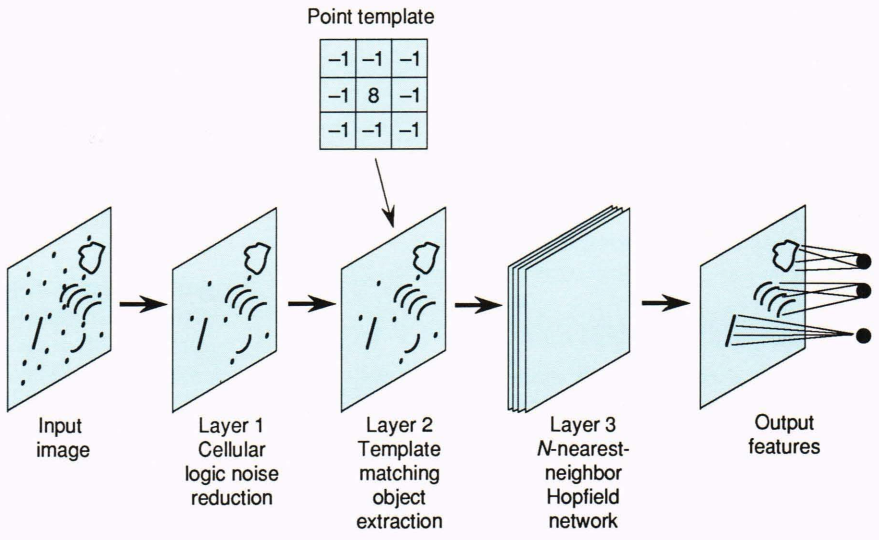

Point template
−1 −1 −1
−1 8 −1
−1 −1 −1
Input image
Layer 1 Cellular logic noise reduction
Layer 2 Template matching object extraction
Layer 3 *N*-nearest-neighbor Hopfield network
Output features

Figure 14. A layered neural-network architecture for extracting image features using both feedforward and feedback connections on the QUEN. Dots represent noise, straight and curved lines represent patterns, and shapes represent point objects.

system. It is similar to systolic arrays such as the WARP machine,[5] but distinctive in its asynchronous linking memories and hardware-implemented data-flow flags. These features extend the capabilities of the QUEN beyond those of systolic processors, providing a higher throughput rate for most algorithms, an expanded range of algorithms that can be implemented, and easier programming.

A wide range of problems has been implemented on the QUEN at JHU/APL and other universities. We have attained respectable improvements in computation speed, ranging from factors of 50 to several hundred in each case.

Because the QUEN is so new, a compiler has yet to be developed, which means that the user/programmer must decompose applications across the array. It is a myth that multiple-instruction, multiple-data machines are impossible to program. A coherent computational model exists for the QUEN in the wavefront computation concept, and a full set of software tools exists under VAX/VMS for assembly, linking, and use of QUEN routines from the C and Fortran languages. The tools are no more difficult to learn than a new word-processor package. The QUEN becomes easy to use once people start thinking in parallel.

REFERENCES

[1] Dolecek, Q. E., *Parallel Processing for VHSIC Systems*, VHSIC Applications Workshop, JHU/APL, pp. 84–112 (1984).

[2] Kung, S. Y., Lo, S. C., Jean, S. N., and Hwang, J. N., "Wavefront Array Processors—Concept to Implementation," *Computer* **20**, 18–33 (1987).

[3] Hockney, R. W., and Jesshope, C. R., *Parallel Computers*, Adam Hilger Ltd., Bristol, England (1986).

[4] Roth, M. W., "Neural-Network Technology and Its Applications," *Johns Hopkins APL Tech. Dig.* **9**, 242–251 (1988).

[5] Annaratone, M., Arnould, E., Gross, T., Kung, H. T., Lam, M., et al., "The WARP Computer: Architecture, Implementation, and Performance," *IEEE Trans. Comput.* **C-36**, 1523–1537 (1987).

THE AUTHOR

QUENTIN E. DOLECEK was born in Sioux Falls, S.D., in 1940. He received a B.S. degree in electronics from the University of Maryland in 1963. After two years in the Navy, he received an M.S. in acoustics and a D.Sc. in electronics from the Catholic University of America, in 1970 and 1980, respectively. Dr. Dolecek began his professional career in 1958 at the Naval Ship Research and Development Center, where he participated in early submarine radiated noise measurements. In 1970, he moved to the Submarine Self-Noise Group at the Center, where he designed high-performance signal-processing systems for at-sea measurements. He joined APL in 1980, where he is a specialist in signal processing, systems analysis, and computer graphics.

Dr. Dolecek has been principal investigator and co-investigator on a variety of Defense Department programs. He participated in the Defense Department's Very-High-Speed Integrated Circuit (VHSIC) program, both in the system design and VHSIC insertion activities. He was principal investigator for high-speed sonar processing algorithms, high-speed bus structures for multiprocessing, and the memory-linked wavefront array processor (MWAP). Inventor of the MWAP technology, he led APL's development of a prototype machine and transfer of the technology to industry. His research activities are in digital design and parallel algorithms.

Dr. Dolecek is a member of APL's Principal Professional Staff and is a section supervisor in the Signal Processing Group of the Strategic Systems Department.