J. ROBERT BUCHANAN

AUTOMATIC TARGET RECOGNITION ON THE CONNECTION MACHINE

Automatic target recognition (ATR) is a computationally intensive problem that benefits from the abilities of the Connection Machine (CM), a massively parallel computer used for data-level parallel computing. The large computational resources of the CM can efficiently handle an approach to ATR that uses parallel stereo-matching and neural-network algorithms. Such an approach shows promise as an ATR system of satisfactory performance.

INTRODUCTION

This article describes a two-part approach to automatic target recognition (ATR). The first part uses stereo-processing algorithms to produce a three-dimensional representation of a target, and the second uses the associative memory and pattern-recognition properties of neural networks to identify or classify targets.[1] This approach shows much promise because three-dimensional representations of objects capture more useful information and require less storage capacity than a series of two-dimensional representations. The three-dimensional representations are incomplete and somewhat distorted, however, because of hidden surfaces, occluded details, and noise in the input image data. Still, classifiers based on neural networks provide some robustness and fault tolerance to compensate for some of the defects of the three-dimensional representations. Together, the two technologies show promise as an ATR system of satisfactory performance.

The ideas for using stereo processing with neural networks for ATR came from Michael W. Roth and Robert L. Kulp of APL. Roth is the principal investigator of an independent research and development project aimed at performing ATR, and Kulp and J. Robert Buchanan are co-investigators. Although large serial-processing computers cannot perform the computations required for this approach within the time constraints involved, large parallel-processing computers such as the Connection Machine (CM) can.

Parallel processing is the application of multiple processors to the execution of an algorithm.[2] The combined data-processing abilities of multiple processors can significantly shorten the execution time of many programs. The improvement in execution time for parallel processors over serial processors is a function of the architecture of the computer and the parallelism inherent in the events and data structures of the algorithm.

THE ALGORITHMS

Stereo Matching

Digital images are usually rectangular arrangements of picture elements, commonly called pixels. Each pixel encodes some information about the objects represented in the image, and the encoded information is usually the intensity of reflected radiation. One goal of stereo processing is the computation of an elevation map from the intensity information in a stereo pair, which is a pair of digitized views of the same scene captured at the same range but from different horizontal positions. From the differing information in each view, a three-dimensional representation of the scene can be constructed. One useful representation is an elevation map, which indicates the elevations of objects and their components in the scene. The stereo-processing algorithm derives elevation information from the apparent shifting of objects in the scene, depending on whether they are viewed from one horizontal position or the other. The perceived shift of an image component is related to the elevation of the component; higher features are shifted more than lower features. Features of objects in the image are detected by their edges. A discontinuity in the intensity of reflected radiation usually occurs at the edges of objects or where objects partially occlude one another. The following steps are used to compute an elevation map:[3,4]

1. Detect the edges in each view of the stereo pair, generating an edge map of each view.
2. While holding one edge map stationary (assume it is the edge map generated from the image data collected by the left eye), slide the other edge map over the stationary edge map. At each shift note the positions at which edges match in the two edge maps. A shift value at which many edges match in a neighborhood is likely to be the appropriate elevation for the neighborhood. The number of edge matches in a neighborhood is the local support score for the current shift value in that neighborhood.
3. Determine the local support score for each shift value by counting the number of edge matches in a neighborhood around each edge match.
4. For each pixel in the image at which an edge match was detected, the shift value with the maximum local support score is defined to be the appropriate elevation.
5. The pixel positions at which no edge matches were

detected derive an elevation from their neighbors' elevations through an interpolation scheme.

Alternative algorithms can be used to compute an elevation map. In practice, additional processing steps are required to reduce noise, enhance contrast, and perform other image manipulations.

Back Propagation

Neural networks consist of a collection of neurons interconnected in a topology via weighted links and an algorithm for changing the link weights to encode information in the network. The neurons are often called "units," which can be classified according to whether they receive stimuli from their environment (input units), express the network's response to stimulation (output units), or connect only to other units (hidden units). The topology of the links between units affects the learning abilities of networks. Each unit sums its inputs, which may come from the outside world in the case of input units or from other units in the case of hidden and output units, and applies an "activation function" to generate an output that is propagated on the unit's outbound links. The algorithm used to update link weights and encode information is called the "learning algorithm." Neural networks are further discussed in the articles by Vincent Sigillito elsewhere in this issue.

Back propagation is one of several neural-network learning algorithms; it is used to train neural networks with one or more layers of hidden units.[5] Hidden units have no direct connection to the outside world. All units compute an activation, which is a function of the sum of the units' inputs. A unit's input is the weighted sum of the outputs of the units to which the unit is linked. The use of hidden units and nonlinear activation functions overcomes the limitations of other neural-network learning paradigms. The back-propagation learning algorithm iteratively minimizes the mean square error between the actual output of the output layer and the desired output.[2,5]

The use of nonlinear activation functions is important in multilayer networks because a single-layer network with linear activation functions and appropriately chosen weights can perform the same calculations as any multilayer network with linear activation functions. The nonlinear activation function most commonly used is the sigmoid function:

$$f(x) = \frac{1}{(1 + e^{-x})} \quad . \tag{1}$$

Layers of hidden units provide the necessary freedom for a neural network to develop an internal representation of the mapping between its inputs and its desired outputs.

Figure 1 shows a section of a neural network. The back-propagation learning algorithm is described as follows:

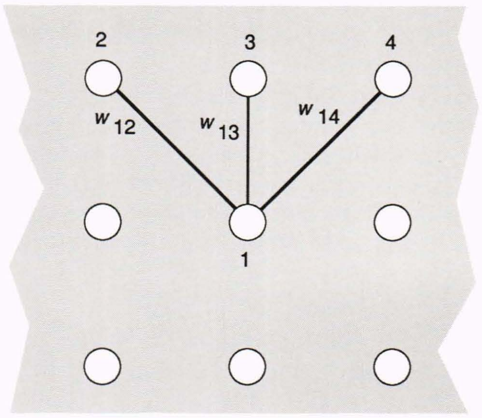


Figure 1. A section of a multilayer neural network. Units are depicted as circles, and weights are shown as lines connecting units; w_{ij} is the weight connecting unit i to unit j.

1. Initialize all link weights to random values.
2. Present an input pattern to the units of the input layer and the desired output pattern to the units of the output layer.
3. Compute the actual outputs by using the sigmoid function given in Equation 1.
4. Starting at the output nodes and working backward to the first hidden layer, adjust the weights according to the following equation:

$$w_{ij}(t + 1) = w_{ij}(t) + n\partial_j x_i \ , \tag{2}$$

where

$w_{ij}(t)$ = weight connecting the output of unit i to the input of unit j at time t,
n = learning rate (a constant usually between 0.25 and 0.33),
∂_j = error attributable to w_{ij},
x_i = output of unit i.

The error for output units is given by

$$\partial_j = y_j(1 - y_j)(d_j - y_j) \ , \tag{3}$$

where y_j is the actual output of the unit and d_j is the desired output. If a unit is a hidden unit, the equation is

$$\partial_j = x_j(1 - x_j)\Sigma_k \partial_k w_{jk} \ , \tag{4}$$

where the difference between the desired output and the actual output is computed as the weighted sum of the error terms from all the units in the next layer to which unit j is connected. Activations propagate forward from the input to the output layer while errors propagate backward from the output layer to the input layer. Steps 2 through 4 are repeated until the values of the weights converge.

Images and neural networks are both parallel systems, but they exhibit different types of parallelism. An image is a rectangular data structure of independent pixels. Few of the operations needed to implement the stereo-matching algorithm require interaction other than that between nearest-neighbor pixels. Most of the operations treat each pixel independently. In contrast, a neural network consists of an arbitrarily connected set of units, and nonlocal communication between units is required. The units are organized into layers, and slightly different operations are performed at each layer. A parallel processor able to calculate rapidly for both parts of this approach to ATR would require many processing elements ("massive parallelism" to handle all the pixels in an image simultaneously) and a flexible interprocessor communications network (nearest-neighbor grid-pattern communications for image processing and arbitrary pattern communications for neural networks). The CM provides these features.

Connection Machine Architecture

The CM is a massively parallel computer used for data-level parallel computing.[6] A characteristic used to describe many parallel-processor architectures is the number of processing elements in the design. The CM is called massively parallel because it may contain 65,536 physical processing elements. It is an example of single-instruction multiple-data design architecture, which is the label given to vector and parallel processors that operate by executing a single instruction stream on multiple operands.

The CM is simple, regular, and scalable, and may be configured with 16K, 32K, or 64K physical processors (where K = 1024). It is an attached processing device connected to a serial computer called the front-end processor, which broadcasts an instruction stream to all the CM processors simultaneously. The single-instruction multiple-data design of the CM dictates that all of its processors execute the same instruction simultaneously. Up to four front-end processors connect to the CM. Front-end processors require a special interface circuit board to connect to the CM through the NEXUS, which is a software switch that attaches the appropriate CM processor resources at the request of a front-end processor. The front-end processors may attach a subset of the physical processors rather than all of the processors. Figure 2 is a block diagram showing the main components of the CM.

Each physical processor of the CM has 64 Kb of memory (a CM with 64K processors has 512 MB of memory). The processors implement a bit-serial design: they operate by reading a single bit of each of the current instruction's operands, computing a single-bit result, and writing the result to memory. Operations on data structures occupying more than 1 b of memory are carried out as a series of single-bit operations. Thus, a 32-b addition is executed as a series of 32 single-bit operations. This manner of execution does not burden programmers because programming language statements, even at the relatively low level of the CM parallel instruction set, are automatically broken into bit-serial operations by the CM microcode.

A CM with 64K processors operating on 32-b operands executes the equivalent of 10 billion operations per second, which is equivalent to the combined performance of approximately 10,000 VAX 11/780 minicomputers. The performance of a CM on floating-point operations, however, suffers from the bit-serial design. A floating-point operation might expand into more than a thousand bit-serial operations requiring significant time for completion. Because floating-point arithmetic is fundamentally

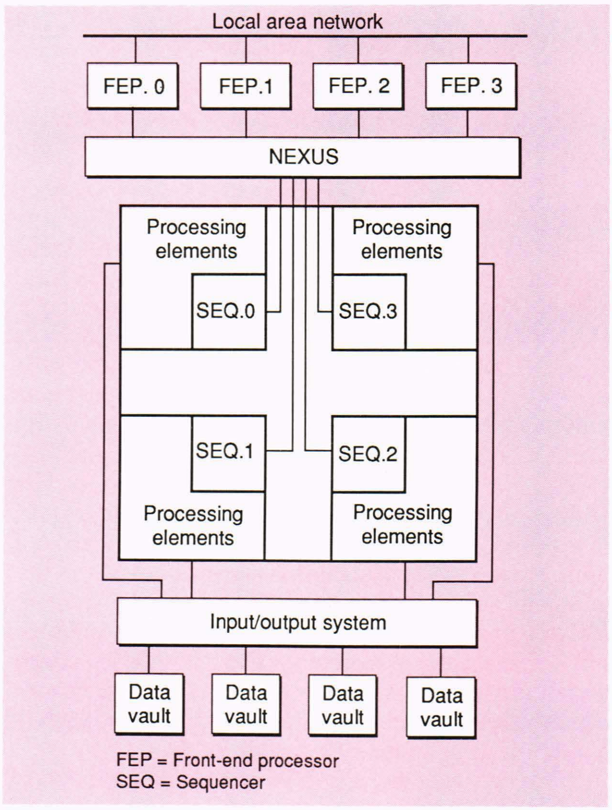


Figure 2. Block diagram of the CM. NEXUS is a software switch that attaches the appropriate CM processor resources at the request of a front-end processor.

important in scientific and engineering computations, floating-point coprocessors are provided (one for every 32 bit-serial processors) to support single-precision floating-point operations. Floating-point coprocessors able to handle double- and single-precision floating-point formats are under development. With the current floating-point coprocessors, a CM can execute 4 billion floating-point operations per second on single-precision data. Computational throughput is one of the strengths of the CM.

Another strength of the CM is its interprocessor communications. Two styles of communications are provided: an n-dimensional grid and a hypercube routing network called the router network. The n-dimensional grid communications, sometimes called NEWS grid (for north, east, west, and south), are optimized for nearest-neighbor communications with a regular pattern. Use of the NEWS grid for communications requires computation of the axis or axes of the grid along which data communications will take place, whereas use of the hypercube router network requires only that the address of the destination processor be computed; the path used to route data traffic to the destination is computed by special processors called "routers" on each CM microchip. The routers operate a packet-switched network responsible for accepting message traffic from processing elements and delivering that traffic to its destination.

The wiring pattern is a 12-dimensional hypercube; each router is connected to other routers by 12 wires, which allow bit-serial communications. Thus, each router is separated by no more than 12 wires from any other router. Hypercubes have the topological property that many other network designs can be mapped onto them. For example, hypercubes of lower dimension, as well as rings and trees, can be mapped onto hypercubes of higher dimension. This property adds to the flexibility of the CM interprocessor communications paradigms. The router network can handle 3 billion bits of message traffic per second.

Use of a two-dimensional communications grid is fundamental to the stereo-matching algorithm to be described later, because images are usually two-dimensional data structures. An arbitrary interprocessor communications network is important to the neural-network algorithm because the units of a neural network are randomly interconnected. Units do not necessarily have the same number of connections.

Another strength of the CM design is its use of virtual processors, which exist when a physical processor subdivides its 64 Kb of memory and repeats the same instruction on data in each memory subdivision. The ratio of virtual processors to physical processors is known as the VP ratio, currently restricted to powers of 2, but the restriction may be removed in the future. Although the memory of each virtual processor is only 64 Kb divided by the VP ratio, the slowdown in execution of instructions is sublinear. This benefit arises from the physical processor decoding the incoming instruction once and then amortizing the cost of that decoding over multiple executions. For a VP ratio of 1, each decoded instruction is executed only once. Higher VP ratios are beneficial in that the incoming instruction for each virtual processor does not have to be decoded. The peak performance of the CM is achieved when using high VP ratios.[7]

The virtual processor feature eases the job of the programmer by allowing the CM to effectively change its number of processors to fit a problem. The stereo-matching algorithm can run on 8K physical processors but uses images digitized as 128 by 128 pixels (16K pixels, a VP ratio of 2), 256 by 256 pixels (64K pixels, a VP ratio of 8), 512 by 512 pixels (256K pixels, a VP ratio of 32), or any higher resolution as long as each virtual processor has enough memory to hold all the data associated with each pixel. The virtual processor feature allows the various image resolutions to be processed without changes to the stereo-matching algorithm (no changes to source code or object code are necessary). The neural-network algorithm may not require that the number of processors be a power of 2. For small networks, many thousands of processors may be left idle, whereas for large networks, the CM can increase its VP ratio until enough virtual processors are available to process the neural network.

In summary, the strengths of the CM are its high computational throughput, the high bandwidth of its interprocessor communications network, the flexibility of its interprocessor communications network, and its virtual processor feature. The CM provides a suitable base on which to experiment with problems too large to be easily implemented on serial processors, but it does require that parallel approaches to algorithms be developed. In many cases, parallel algorithms are easy to state. The greatest challenge is for algorithm designers to abandon the serial thinking that is second nature to them because of years of programming serial machines.

Programming Model

The programming languages of the CM reflect its hardware architecture. All the CM processors execute the same instruction at the same time. The general approach to programming the computer is to distribute homogeneous data across all processors so that each processor has data on which it can operate in parallel. Once data are distributed, a traditional serial program that operates on the data in one processor is written. Programmers have no need to program each processor individually or to program synchronization code to keep all processors in step, because a single-instruction stream generated on the front-end processor is broadcast to all processors simultaneously.

The CM is programmed in parallel extensions to some familiar serial languages. Parallel versions of Common Lisp, C++, and Fortran exist (they are called *Lisp, C*, and CM Fortran, respectively).[8] Serial programs can still be expressed in these languages. The languages and their compilers do not convert serial code into parallel code for the CM; the responsibility for generating parallel code lies with the programmer, not the computer or its code development tools. In contrast, programming languages such as Vast Fortran are able to analyze serial "DO" loops and produce vector operations when permissible.[9] Thus, to produce parallel code on the CM, a programmer must explicitly use the parallel extensions to the serial syntax. For data-parallel programs (programs in which the same operation is applied to a large amount of data), programming in a parallel language can be easier than programming in a serial one.

The three high-level languages of the CM—*Lisp, C*, and CM Fortran—extend the meaning and operation of serial functions such as addition or multiplication to work on parallel data or combinations of scalar and parallel data. The term "scalar" is used to refer to data stored on the front-end processor. It could be a single datum such as a floating-point number or character, or it could be an array of numbers stored on the front end. On the other hand, parallel data or "pvars" (short for "parallel variables") are distributed among the processing elements in the CM. Any statements that mix scalars and pvars promote scalars to pvars by copying the scalar value to temporary storage in each virtual processor and then performing a parallel operation.

The best way to distribute data throughout the CM is not always obvious. One of the challenges of working with the CM is determining ways of distributing problems to maximize the exposure of the data to the virtual processors.

IMPLEMENTED ALGORITHMS

Stereo Matching

Because the images of interest are square matrices 512 pixels on a side and most of the operations necessary to do stereo matching require only nearest-neighbor communications, it is convenient to map the image onto the CM as if the virtual processors were arranged on a square grid 512 processors on a side. The parallel algorithm will require 262,144 virtual processors, each dedicated to a single pixel. The stereo-matching algorithm described in this section performs the same operation on each pixel. The descriptions of the four steps of the algorithm are specific to performing stereo matching in a data-parallel style. On a serial processor these operations would be implemented as nested DO loops, which iteratively operate on each pixel. The time to execution of these loops would be $O(512^2)$. [$O(n)$ means proportional to n.] On the CM these same operations are computed in $O(1)$ time, 5 orders of magnitude faster.

The algorithm steps and corresponding descriptions are given as follows:

1. Detect the edges in each view of the stereo pair.

The Canny edge-detection algorithm is useful for finding edges in noisy images. It smoothes a raw image by convolving it with a Gaussian filter several pixels wide. The gradient of the intensity changes of the filtered image is then computed. Edges are assigned at pixel locations where the intensity gradient is above a threshold computed from a measurement of the noise present in the image.[10] This algorithm requires computation on data within each virtual processor and some nearest-neighbor interprocessor communications. The output of the Canny edge detector is a 1 stored in a field in each virtual processor whose position corresponds to an edge in an image or a 0 where no edge is present.

2. While holding one edge map stationary (assume it is the edge map generated from the image data collected by the left eye), slide the other edge map over the stationary edge map. At each shift note the positions at which edges match in the two edge maps.

This process iterates for some programmer-specified number of steps. The computations performed at each step are performed in parallel. Since a 1 signifies the presence of an edge, a logical "AND" of the left edges and the shifted right edges will indicate edge alignments. The result of this step is a field in each processor n bits wide, where each bit position containing a 1 signifies edge alignment at that shift. This process requires n nearest-neighbor interprocessor communications across the square grid and n logical AND operations on fields 1 b long. Because the algorithm is implemented by using a virtual processor ratio greater than 1, many of these nearest-neighbor communications correspond to the movement of data within a single physical processor.

3. Determine the local support score for each shift value by counting the number of edge matches in a neighborhood around each edge match.

To determine the appropriate shift for each pixel, it is necessary to determine how well edges match over some small region of the image. Continuity and smoothness constraints must be met. Regions in the image where many edges align are likely to be part of the same image components and at nearly the same elevation. Each virtual processor counts the number of edge alignments in a small square around itself at each shift of the edge maps. The result of this step is an array within each virtual processor. Each array element indicates the number of edge matches found in a square about a pixel at each shift value.

4. For each pixel in the image at which an edge match was detected, the shift value with the maximum local support score is defined to be the appropriate elevation.

Each virtual processor examines the array of edge-alignment scores and selects the greatest value. Consistency requires that this value be related to the elevation of the pixel in the unprocessed image.

Back Propagation

The heart of any back-propagation implementation is computing the sum of the weighted inputs to a unit and the sum of the weighted error terms to a unit. Because units may each have different numbers of input and output links, an implementation that iterates over the links to a unit must do some bookkeeping on the number of links for each unit. This requirement adds complexity to the implementation. The CM provides parallel scan instructions that execute efficiently and liberate the parallel back-propagation implementation from treating units differently on the basis of their number of input and output links.

Scans and their effects will be familiar to users of Kenneth Iverson's A Programming Language.[11] A scan allows a binary associative operation to be applied to all the initial subsequences of a vector. The result is a vector whose elements are the result of applying the operation to the first element of the input vector, the result of applying the operation to the first two elements of the input vector, and so on. The result of a plus-scan on the vector 1,2,3,4 is 1,3,6,10.

The parallelism of the CM allows scans to be computed in $O(\log n)$ time, where n is the number of virtual processors.[12] Figure 3 illustrates a plus-scan on a hypothetical eight-processor CM. The symbol X represents the pvar being scanned and can have a different value in each processor. The symbol $S(i,j)$ represents the result of the plus-scan, that is, the sum of the X's from processors i to j inclusive. The algorithm can be thought of as a loop that is executed $\log_2$(number of virtual processors) times. In the body of the loop, each virtual processor sends the partial sum of X values it holds to a virtual processor whose hypercube address is a power of 2 greater than its own hypercube address, provided the destination virtual processor of the send operation exists. Because the number of processors is finite, a processor should not send off the hypercube.

At the start of the scan, every virtual processor holds an X value, which can be thought of as the partial sum of one term. For the first step, each processor sends its partial sum to the virtual processor whose hypercube address is $2^0 = 1$ greater than its own. Each virtual processor receiving a partial sum adds it to the partial sum it already holds. For the second step, each processor

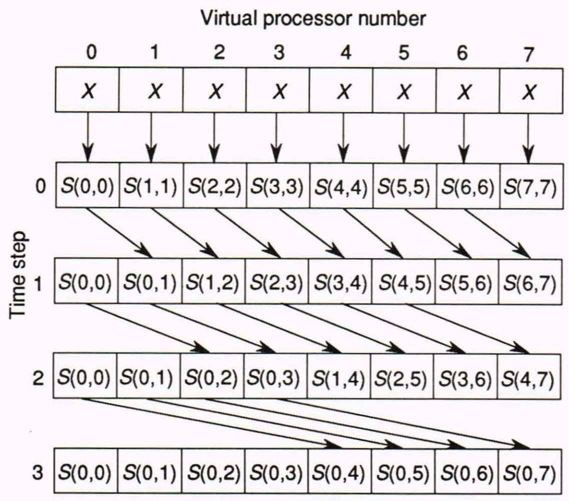

Virtual processor number
0 1 2 3 4 5 6 7
X X X X X X X X
Time step
0 S(0,0) S(1,1) S(2,2) S(3,3) S(4,4) S(5,5) S(6,6) S(7,7)
1 S(0,0) S(0,1) S(1,2) S(2,3) S(3,4) S(4,5) S(5,6) S(6,7)
2 S(0,0) S(0,1) S(0,2) S(0,3) S(1,4) S(2,5) S(3,6) S(4,7)
3 S(0,0) S(0,1) S(0,2) S(0,3) S(0,4) S(0,5) S(0,6) S(0,7)

Figure 3. Graphical representation of a plus-scan operation in $\log_2(n)$ steps, where n is the number of virtual processors. X represents the pvar being scanned and can have a different value in each processor, $S(i,j)$ represents the results of the plus-scan, and the arrows indicate the sources and destinations of the interprocessor communications operations.

sends its partial sum to the virtual processor whose cube address is $2^1 = 2$ greater than its own. Again, each virtual processor receiving a partial sum adds it to the partial sum it already holds. These steps are repeated $\log_2$ (number of virtual processors) times. In the end, each processor holds the sum of its original X value and the sum of all the X values from virtual processors whose hypercube addresses are less than its own. The partial results at each time step are shown in Figure 3. The arrows indicate the sources and destinations of the interprocessor communications operations.

The plus-scan operation is a single program instruction even when programmed in the CM assembler language. Programmers do not program the loop described in the preceding paragraph; instead, the operations take place at the level of the CM microcode.

The scan operation can start from either end of the hypercube addresses (0 to the limit of the number of processors, or the limit of the number of processors to 0), and scanning along axes of the NEWS grid is also allowed. The segmented scan is particularly useful to the back-propagation algorithm. A scan operation can be restarted at various processors in the hypercube, depending on the value of the segment pvar. Whenever the value "TRUE" is stored in a segment pvar, the scan restarts at that processor. Table 1 gives the inputs and results of a segmented scan.

The relevance of segmented scans to back propagation will become apparent. A single virtual processor is used for each unit, and two virtual processors are used for each weight because a weight is simultaneously a factor on an input link for a unit and a factor on an output link for a different unit.[13] The virtual processor feature of the CM should provide enough virtual processors to simulate large neural networks with many units and links.

The units and weights are assigned to virtual processors according to their layer number and whether a weight is an input or output weight. The input layer occupies virtual processors at the low end of the hypercube, then the hidden layer, and then the output layer. Within a layer, a unit is preceded by all of its input weights and followed by all of its output weights. Figure 4 illustrates how a simple network might be mapped. In the figure, fan-in weights refer to the number of links coming into a unit, and fan-out weights refer to the number of links emanating from a unit.

Scans are used to copy the activation value of a unit to its output links, compute the weighted sum of the in-

Table 1. The source pvar, segment pvar, and result pvar of computing a segmented plus-scan.

	Hypercube address							
	0	1	2	3	4	5	6	7
Source pvar	3	6	2	8	2	−1	2	1
Segment pvar	T	F	F	F	T	T	F	T
Result pvar	3	9	11	19	2	−1	1	1

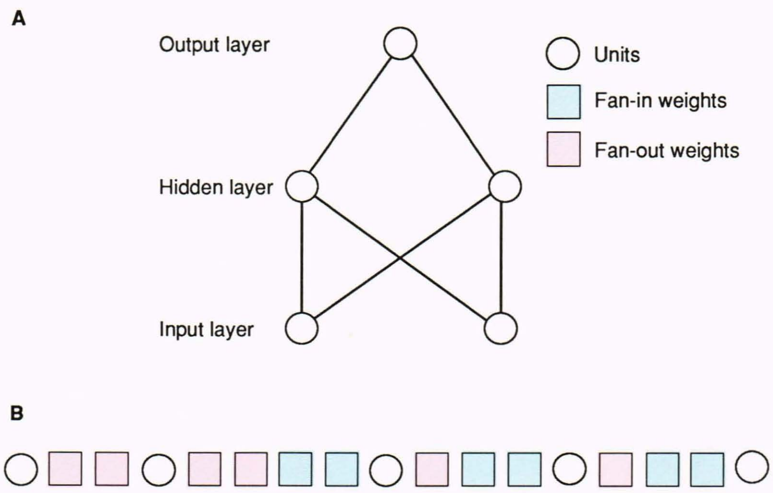


Figure 4. Mapping a neural network onto the CM. A. Block diagram of a simple neural network. B. Illustration of how the network's units and weights are assigned to the virtual processors of the CM; two virtual processors are used for every weight.

puts to a unit, copy the error term from a unit to its input links, and compute the weighted sum of the error terms for a unit. The feedforward phase of a neural-network cycle scans in the forward direction, and the back-propagation phase of the cycle scans in the backward direction. The two phases are otherwise programmed similarly.

To feedforward, each virtual processor representing a unit copies its activation to the virtual processors representing the unit's output link weights. Each virtual processor representing an output link weight also contains the hypercube address of the weight's twin (a virtual processor that represents an input link weight corresponding to the output link weight, since every output link is an input link for some other unit). Each virtual processor representing an output link weight sends the activation value it received via the copy-scan to its twin. Finally, a plus-scan is computed, which sums the products of the input link weights and the activation values previously sent. As a result, each unit has received a new input. The units can calculate their new activation values in parallel.

Figure 5 illustrates the feedforward process. This algorithm has the advantage of simplicity because the bulk of the work is done in only three machine-level instructions. It is divorced from any consideration of the number of fan-in or fan-out weights of a unit. The algorithm also makes efficient use of the hardware because scans are computed rapidly and execution time is proportional to the logarithm of the number of virtual processors. Back propagation is computed similarly, except that the direction of the scan is reversed and the quantities being scanned are the error terms.

The neural network as mapped onto the CM forms a pipeline. For a network consisting of an input layer, one hidden layer, and an output layer, two feedforward cycles are required for an input to propagate its effects to the output layer; the same is true of error terms. The algorithm uses the pipeline to propagate two input patterns and sets of error terms simultaneously.

SUMMARY AND CONCLUSION

Automatic target recognition requires large computational resources and therefore benefits from the computational abilities of parallel-processing computers such as the CM. The CM has a massively parallel computer architecture that has been found to be applicable to a variety of algorithms. Its large number of physical processors and the virtual processor mechanism allow it to match the size of various problems to be solved. In addition, the CM provides two powerful methods of interprocessor communication: a grid-based nearest-neighbor network and a hypercube-based router network. Both styles of communication are sometimes used in solving the same problem.

The CM challenges programmers and algorithm designers to cast problems in parallel terms. Serial thinking imposed by programming on serial hardware and languages must be abandoned in favor of parallel thinking, which more closely describes the true behavior of natural phenomena. Parallel thinking yields new insights into algorithms, and the CM provides a computational engine for exploring algorithms and areas that are impractical on serial computers.

REFERENCES

[1] Roth, M. W., "Survey of Neural Network Technology for Automatic Target Recognition," to be published in *IEEE Trans. Neural Networks* **1** (1989).

[2] Lipovski, G. J., and Miroslaw, M., *Parallel Computing,* John Wiley and Sons, New York, p. 2 (1987).

[3] Drumheller, M., *Connection Machine Stereo Matching,* V86-2, Thinking Machines Corporation, Cambridge, Mass. (Mar 1986).

[4] Drumheller, M., and Poggio, T., "On Parallel Stereo," in *Proc. 1986 IEEE International Conf. on Robotics and Automation*, IEEE Council on Robotics and Automation, San Francisco, pp. 1439-1448 (1986).

[5] Rumelhart, D. E., and McClelland, J. L., eds., in *Parallel Distributed Processing, Explorations in the Microstructure of Cognition*, MIT Press, Cambridge, Mass. (1986).

[6] Hillis, W. D., *The Connection Machine,* MIT Press, Cambridge, Mass. (1985).

[7] Thinking Machines Corporation, *Connection Machine Model CM-2 Technical Summary,* HA87-4, Cambridge, Mass. (1987).

[8] Thinking Machines Corporation, *Introduction to Data Level Parallelism,* TR86-14, Cambridge, Mass. (1986).

[9] Pacific Sierra Research Corporation, *Vast User's Guide*, N-355-V, Los Angeles (1984).

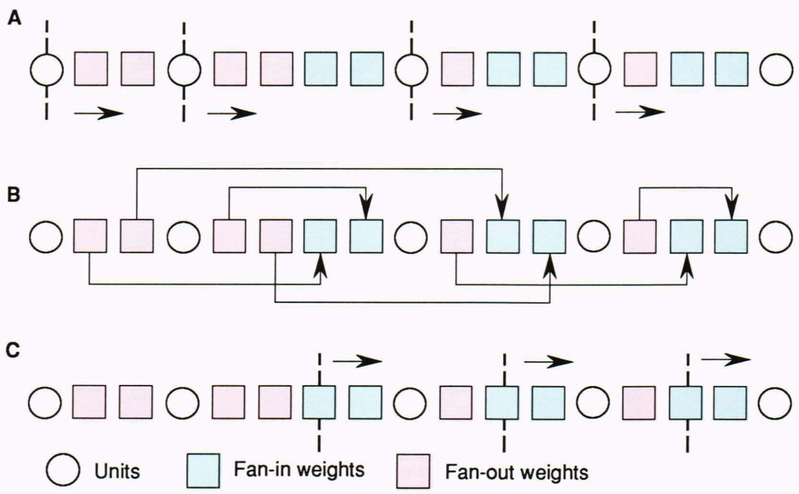


Figure 5. Computation of the parallel feedforward phase in three steps. A. Segmented copy-scan. B. Interprocessor communications. C. Segmented plus-scan. The dashed vertical lines represent the segment boundaries for segmented scan operations, and the arrows represent the direction of scan operations or the sources and destinations of interprocessor communications operations.

[10]Canny, J. F., "Finding Lines and Edges in Images," *MIT AI Memo* 720, MIT Artificial Intelligence Laboratory, Cambridge, Mass. (1983).

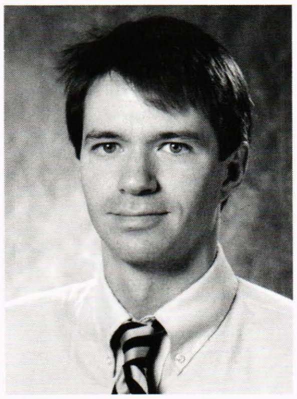
[11]Iverson, K. E., *A Programming Language,* John Wiley and Sons, New York (1962).

[12]Blelloch, G., *Parallel Prefix vs. Concurrent Memory Access,* Thinking Machines Corporation, Cambridge, Mass. (Oct 1986).

[13]Rosenberg, C. R., and Blelloch, G., *An Implementation of Network Learning on the Connection Machine,* Thinking Machines Corporation, Cambridge, Mass. (1986).

ACKNOWLEDGMENTS—The author would like to thank Michael W. Roth and Robert L. Kulp of APL for their ideas on stereo processing and neural networks, which formed the basis of the parallel algorithms described in this article; Henry Dardy of the Naval Research Laboratory (NRL) for permission to use NRL's Connection Machine; Etienne Deprit of NRL for access to his recurrent back-propagation neural-network code (which I modified to perform simple back propagation); Michael Drumheller of the Massachusetts Institute of Technology Artificial Intelligence Laboratory for discussions on programming the parallel stereo algorithms; and Robert Whaley of Thinking Machines Corporation for answering questions about the operation of the CM and help in debugging code.

THE AUTHOR



J. ROBERT BUCHANAN grew up in North Carolina, where he received a B.S. degree in physics from Davidson College in 1983 and an M.S. degree in mathematics from North Carolina State University in 1985. He joined APL in 1986 as a member of the Associate Staff Training Program. As a member of the Computing Systems Group, Mr. Buchanan's research interests include the application of parallel processing to artificial intelligence and scientific and engineering computing. He is a member of the Mathematical Association of America.