A Parallel Neural Processor for Real-Time Applications

TRAINING AND TESTING ARTIFICIAL NEURAL NETWORKS CAN BE CHALLENGING AND TIME-CONSUMING. EXPERIMENTS WITH TWO REAL-TIME APPLICATIONS COMPARED THREE APPROACHES FOR IMPLEMENTING A MULTILAYER PERCEPTRON NEURAL NETWORK. IN BOTH APPLICATIONS, THE SPECIAL-PURPOSE PROCESSOR PERFORMED BEST.

Giovanni Danese
Francesco Leporati
University of Pavia
INFM, Pavia

Stefano Ramat
University of Pavia

Soft computing—genetic algorithms, fuzzy sets, chaos theory, expert systems, and artificial neural networks—provides a simple way to solve complex problems. ANNs, in particular, have stimulated many theoretical studies and experiments.[1] Researchers have proposed several architectures,[2] but the most popular is multilayer perceptron. MLP networks implement a correspondence between input and output vectors through an assigned function with many parameters (weights). Computing these weights is an optimization problem that an analytical approach usually cannot solve. An MLP network can model the unknown function by reducing the optimization problem to a nonlinear problem of dimension equal to the number of ANN parameters.[3]

In practical applications where the training data set is large, execution times on serial machines can be extremely long. Therefore, researchers have initiated several projects to explore parallel architectures for simulating ANNs. These activities involve implementations on general-purpose parallel computers and *neurocomputers* (hardware dedicated to ANN simulations). Examples include ANN simulations on MAS-Par MP-1, Hypercube and Connection machines, and other supercomputers.[4-6] Other groups have designed and built parallel systems based on field-programmable gate arrays, transputers, or digital signal processors (DSPs).[7-9] Several companies have proposed custom-designed VLSI circuits that act like ANN accelerators—for example, CNAPS (Coprocessing Node Architecture for Parallel Systems) from Adaptive Solutions, My-Neupower from Hitachi, and Synapse-1 from Siemens.[10-12] Researchers have also focused on developing faster algorithms.[13]

In the past few years, the Microcomputer Laboratory at the University of Pavia has begun researching hardware solutions for heavy computing problems. This activity involves ANNs used in typical real-time industrial applications, such as signal filtering or compression, recognition, and control. We have implemented these ANNs on either workstations or architectures that integrate existing microcomputers and microprocessors. Here, we compare the performance of three such solutions for two real-time industrial applications and identify the most promising way to train and test an MLP neural network.

0272-1732/02/$17.00 © 2002 IEEE

ANNs and MLP networks

Implementing an ANN is possible using a massively parallel processor that has learning tools for collecting knowledge from experience. In our simulations, we used an MLP network—a multilayer ANN comprising an input layer of sensor units, one or more hidden layers of intermediary nodes, and an output layer of computation nodes. The signal propagates along this network from the input to the output.

An MLP has the following properties:

- Each neuron model presents a smooth, nonlinear output.
- Hidden layers between the input and output layers help the network learn complex problems by progressively identifying meaningful aspects from the input data set.
- The large number of synaptic connections determines the high degree of connectivity; a change in a connection influences the entire population of synapses and their weights.

MLP networks can solve complex problems through a supervised learning process based on back propagation. BP consists of two distinct elaboration phases: one forward and one backward. The forward step applies an input vector (pattern) to the network, and its effect propagates through the layers, yielding a set of values at the output layer as the network's response. The network compares this response to a set of intended target values. The backward step then corrects the synaptic weights by considering the errors between the real and expected outputs.

In a practical application of the BP algorithm, the learning process involves consecutive presentations of a training set to the network; the training set has a variable, random sequence of input patterns. We call one presentation of the entire training set an *epoch*. Learning proceeds, epoch by epoch, until the network's synaptic weights and threshold levels stabilize, and the squared error over the entire set converges to some minimum value. BP usually presents a training set to the network to evaluate the synaptic weights. The network considers as many training examples as possible to achieve good *generalization*. A network generalizes well when it produces a correct output pattern for a new (unused) input data set. Thus, the learning process is essentially a curve-fitting or nonlinear-I/O-mapping problem.

Comparison of three approaches

Scientists typically use one of three implementations to run real-time ANN applications:

- a PC with software tools for analysis and simulation—in this case, a PC equipped with a 400-MHz Intel Pentium II processor with 128 Mbytes of RAM running MathWorks Matlab;
- a digital signal multiprocessor—in this case, the TMS320C80 from Texas Instruments; or
- a special-purpose neural-network processor—in this case, a NeuriCam Totem PCI board with two NC3001 processors.

The last two approaches have an advantage because they can implement high-communication-bound problems that have intrinsic parallelism straight onto the hardware. Such intrinsic parallelism is typical of applications amenable to neural-network solutions. We did not consider parallel supercomputers in these experiments, because their cost and dimensions make them unsuitable for real-time industrial applications.

We compared the performance of these three solutions using two applications. Results show that, for both applications, the special-purpose processor is the most promising solution. With relatively few neurons, this ANN-dedicated hardware chip outperformed the other two implementations. Moreover, results show that, due to their complex programmability and quick obsolescence rate, parallel general-purpose chips are not suitable for these applications.

PC running software tools

A commonly available software tool, Matlab (from The Mathworks) has a built-in set of functions and a graphical user interface for the design, implementation, visualization, and simulation of neural networks. In our experiments, we use the basic package (version 5.1) and an add-on—Neural Network Toolbox,

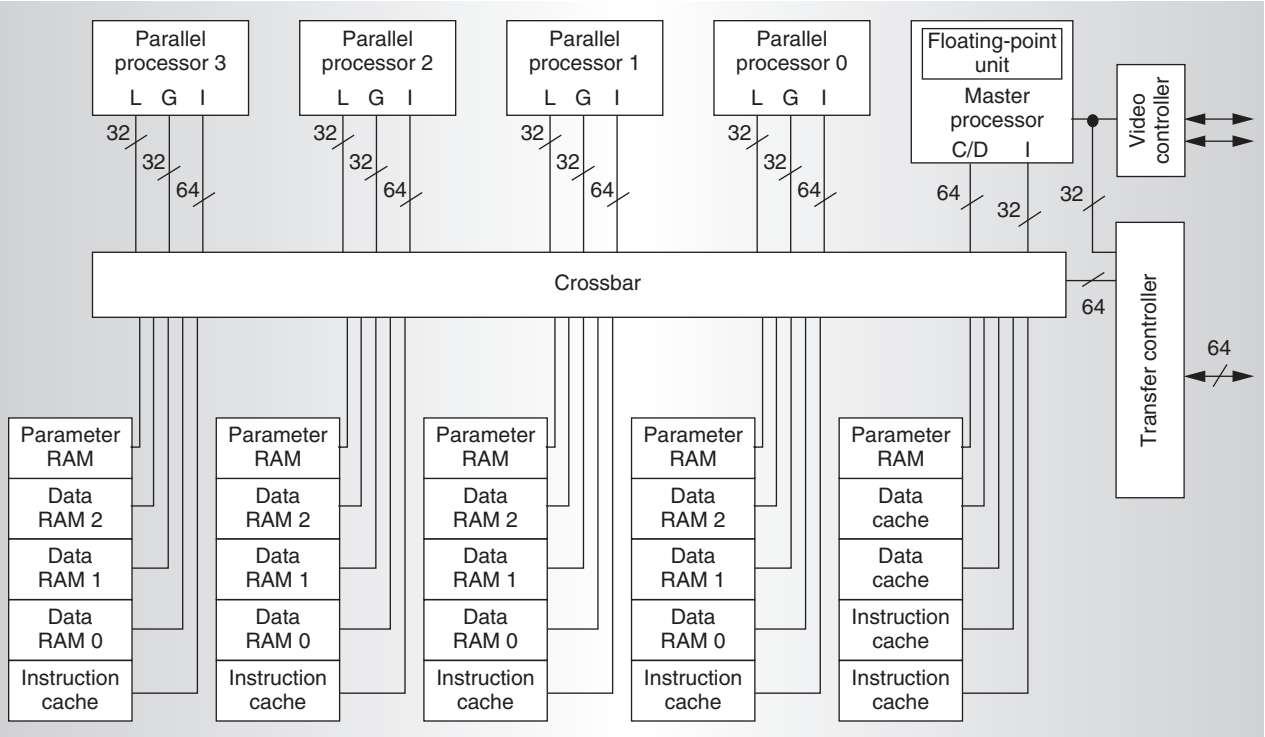

Figure 1. TMS320C80 block diagram.

version 2.0—to program, train, and simulate an ANN for a biomedical-data fitting problem. The ANN was a specific element of a mathematical model developed in Simulink 2.1 (another Matlab add-on), which represented the signal processing for generating eye movements, given the movement of the head.

Digital signal multiprocessor

The TMS320C80, depicted in Figure 1, has one master and four DSP parallel processors. The chip also has 50 Kbytes of SRAM and a direct memory access (DMA) transfer controller (TC) to interface with up to 4 Gbytes of external memory or to control data transfer between two regions of internal memory. The master processor is a 32-bit RISC processor with a floating-point unit. The parallel processors are 32-bit DSPs, featuring hardware solutions and software libraries to speed up image processing or applications with manipulations of data structured in bit fields. The TMS320C80 can reach a peak performance of 2 billion operations per second.

The master processor has several pipelines, including those for instruction execution, floating-point multiplication, and floating-point addition. Using these pipelines, the master processor can execute a single-precision multiplication or double-precision sum in one clock cycle. Special parallel instructions let the master processor reach a 100-Mflops peak performance. The master processor also has two 4-Kbyte instruction caches and one 4-Kbyte data cache.

The parallel processors provide most of the TMS320C80's computing power because each can use a 64-bit instruction word specifying different parallel instructions, independently controlling the data unit and the two address units. Thus, the processing hardware is directly accessible, avoiding the need for an internal microcode unit to translate instructions into corresponding event sequences. Every parallel processor devotes 2 Kbytes of RAM to storing interrupt vectors and TC parameters. These parallel processors also share their 8-Kbyte data RAM blocks with one another, the master processor, and the TC. In contrast, the master processor's memory is accessible only to the master processor itself and the TC. Each parallel processor also has a 2-Kbyte instruction cache.

The TMS320C80 also contains a video controller unit to interface with an image acquisi-

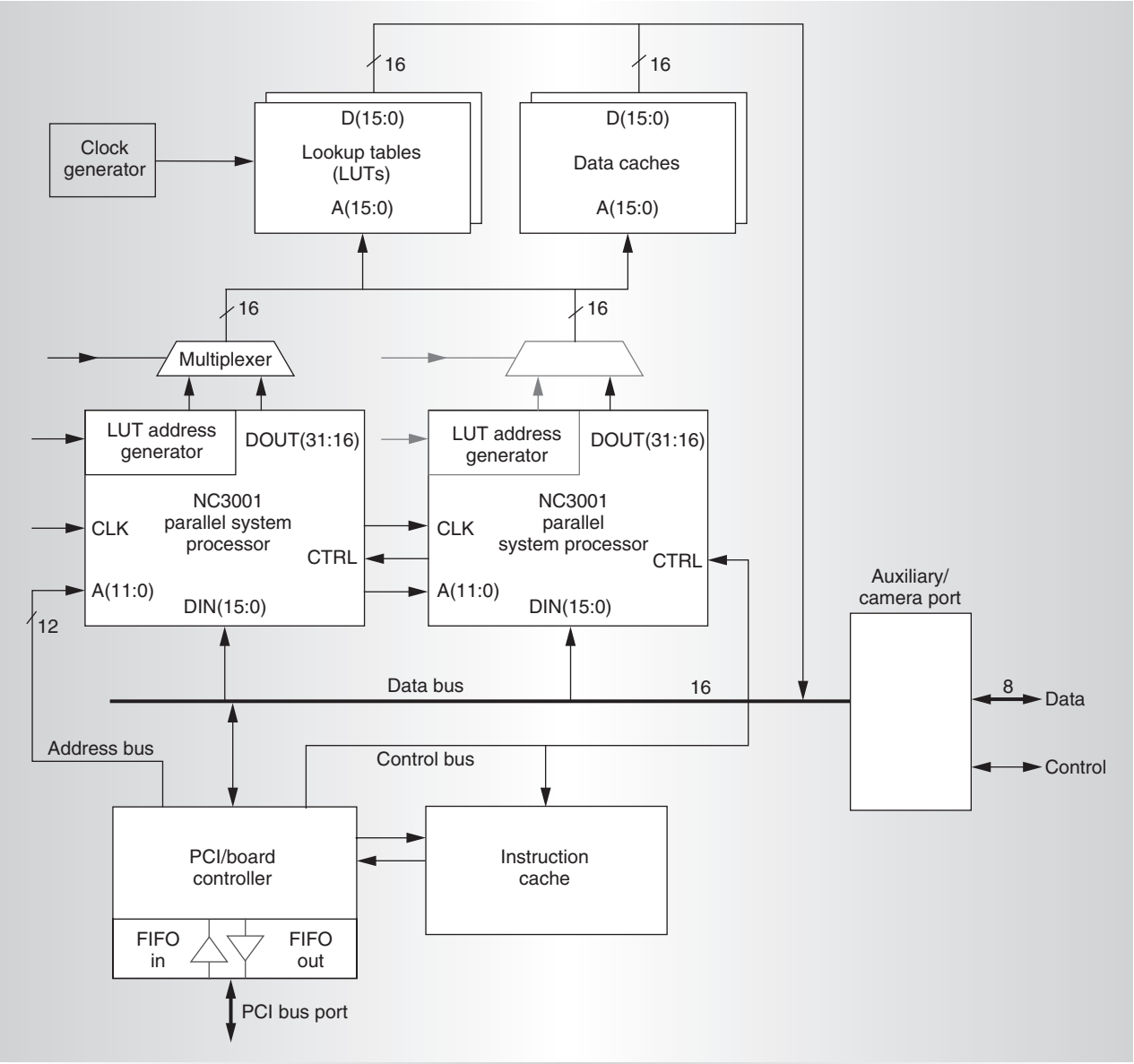

Figure 2. Block diagram of the Totem peripheral-component-interconnect (PCI) board.

tion system or to simultaneously control two frame systems. Processors communicate via a high-speed crossbar network, allowing multiple simultaneous accesses to RAM memory. The crossbar network automatically connects a processor to each addressed RAM, with a maximum bandwidth of 2.4 Gbytes/s. The software development environment includes a compilation and execution package, and a parallel debugger. The latter provides users with a direct interface with the hardware via

- a parallel-debugging manager,
- a command shell that communicates with single debugging processes running on each processor, and
- a Windows-based user interface with different display modes.

The overall TMS320C80 system rests on a board inside a PC and can run under different systems (Windows, Linux, and so on).

Special-purpose processor

The NeuriCam Totem PCI board, depicted in Figure 2, is a neural accelerator for addi-

Reactive tabu search

The tabu search (TS) is an efficient technique for combinatorial optimization. It combines a hill-climbing search strategy using a set of elementary moves (changing one bit at a time in the binary representation of each weight and then evaluating the value of a proper cost function) with a heuristic approach. The latter prevents local minima from blocking the search. TS forbids, for a fixed number of iterations, the use of the inverses of recently used moves. The length of the prohibition period is the *tabu list size*; the forbidden moves essentially enter a tabu list in typical first-in, first-out order. Of course, the choice of the tabu list's size is critical and can significantly affect the search's performance and effectiveness.

To overcome this problem, the reactive tabu search (RTS) algorithm adapts the fixed list's size to the characteristics of the optimization problem. The algorithm records all the configurations considered during the search and checks them for repetitions—thus introducing an adaptable reaction approach along with a diversification method. The adaptable reaction approach increases the list size when local minima cause repetitions in all regions of the search space. The diversification method responds to evidence that the search is trapped in a limited portion of the configuration space without the appearance of the typical periodic behavior caused by a local minimum. Those in the dynamic-systems field would use *chaotic attractor* to describe this situation.

For ANNs, RTS transforms the minimization of the error between the real and expected network outputs into a combinatorial optimization problem easily parallelizable on a multiprocessor system. RTS represents the synaptic weights like bit strings through Gray codes, and represents set μ_i of elementary movements by changing one bit at a time—that is, the ith string bit. These movements define a local search trajectory along the points of a hypercube into which each step is accepted or rejected depending on whether or not it minimizes cost function E. This cost function is the sum of the squared differences between the network's present and desired outputs.

With the codification chosen, achieving the values $n+1$ and $n-1$ near n requires changing only one bit of the code expressing n. Exploring the search trajectory thus corresponds to considering n's neighboring points. This would not be possible if the representation were binary; for example, 1000 follows 0111, but more than one basic move is necessary to perform the transformation. Now the search can easily explore all the neighboring points to find better values of the cost function. Therefore, a search based on this representation can reproduce a binary form of steepest descent faster than back propagation (BP), because it does not have to calculate derivatives and involves only feed-forward steps executed repeatedly until the cost function settles to an asymptotic value.

For more information on RTS, please see the article by Battiti and Tecchiolli.[1]

Reference

1. R. Battiti and G. Tecchiolli, "Training Neural Nets with the Reactive Tabu Search," *IEEE Trans. Neural Networks*, vol. 6, no. 5, Sept. 1995, pp. 1185-1200.

tion to PCs and workstations as a coprocessor. This coprocessor interacts with the main processor in the host system through standard buses. Software tools, including interface libraries and a graphical development system, are available for most popular computer platforms (Windows 98 and NT, DOS, Linux, and Unix).

The Totem PCI board uses two NC3001 ICs. Each NC3001 has several processors (artificial neurons) operating concurrently; their connections mimic the human brain's operation. The NC 3001's performance comes from two important factors: dedicated hardware that implements the neural network and a very efficient training algorithm. The chip is a parallel VLSI computing device with simple fixed-point processing units and an on-chip weight memory optimized for computing MLP-based neural networks, for implementation of the reactive tabu search (RTS) learning algorithm, which avoids the local-minima problem faced by pure-BP approaches and can execute quickly on fixed-point units.[14] RTS requires no transfer function derivatives and is suited for VLSI implementation. For a detailed explanation, see the "Reactive tabu search" sidebar.

The main characteristics of the NC3001 include the following:

- A pipelined data stream and single-instruction, multiple-data (SIMD) architecture executes multiply-accumulate (MAC) operations on 32 processing units (neurons) in parallel and within a single clock cycle.
- Thirty-two fixed-point MAC units operate in parallel and as part of a three-stage pipeline.
- Limited word width—16-bit data, 8-bit weights, and 32-bit results—ensures an economical layout. RTS optimizes word widths for learning. The broadcast bus' 16-bit width can represent signals from transducers and intermediate results between layers. The 8-bit memory word width can handle many classification tasks. The output channel's 32-bit word width permits high accumulation capacity.
- Designers organized the 64-Kbit internal dynamic RAM as 32 blocks of 2K × 8 bits for weight storage that is strictly coupled with neurons. Alternatively, rather than assigning memory to a single neuron, the internal system software can partition it among several neurons to implement multilayer networks with a single chip.
- A 40-MHz clock enables 1.2 billion MAC operations per second. In less than 2 μs, the chip can evaluate a multilayer

perceptron with a $16 \times 16 \times 1$ topology. Higher performance is possible by connecting up to four chips per network level to implement neurons with up to 256 inputs.
- Support circuitry enables external look-up tables (LUTs) to implement activation functions.

Each neuron contains a MAC unit and an output register to allow straightforward implementation of multilayer networks. The global bandwidth required to feed the weights to the MAC operations during training is nearly N I/O operations per accumulation cycle (N is the number of neurons), potentially causing a serious I/O bottleneck by using external memory with such highly parallel architectures. The NC3001 solves this problem by using localized internal memory to store weights. This enables a high bandwidth between MAC operations and memory at the expense of increased silicon area. Other system circuits, such as the host processor, handle the infrequent calculation of weight changes during the learning phase.

Application 1: Classification of 2D space points

Consider a simple classification problem solved through a neural-network approach. Figure 3 shows a region of the *xy* plane with geometric figures A, B, and C. Figures A and B each represent 25 percent of the entire region, and C represents 10 percent. D is 40 percent of the region, and represents the portion not encompassed within A, B, or C.

The problem involves selecting a point in the plane (the input) and deciding to which figure within the region it belongs. We began by selecting random pairs of real numbers, corresponding to coordinates *x* and *y* of points within the region. To each pair of coordinate values we also added a third value, a tag, to indicate the geometric figure associated with the coordinates. We built two sets of points: one for training and one for testing the trained network.

Training and generalization setup

We presented the training input patterns (*x-y* pairs) to the network in random order for successive epochs. After each iteration, we evaluated the network's output and back-propagated the error through the network by updating the weights. At the end of each epoch, a calculation of the average squared error served as an indicator of whether to continue the learning process. Once training ended, we verified the generalization ability of the network by considering how it classified patterns not used in the previous phase.

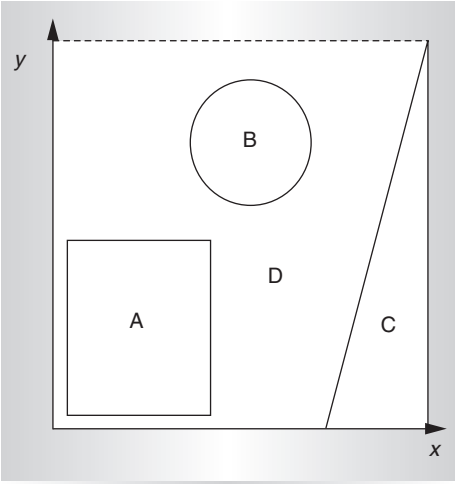

Figure 3. Portions of 2D space considered in a simple classification problem.

The implemented ANN had an input layer with two inputs (the coordinates of the points to be classified), a hidden layer with 16 neurons, and an output layer with four neurons (one for each geometric figure in the *x-y* space). The exact output was a simple 1 for the neuron (indicated by the tag) and three 0s for the others; however, the classification is acceptable when the neuron with the highest output is the one indicated by the tag. The activation function was a nonlinear sigmoid curve.

We used the generalized delta rule to upgrade the weights by changing learning rate η and momentum α. A small η yields a slow convergence but identifies a deeper minimum than the one corresponding to a large η. Moreover, whereas a small η and large α yields a high convergence speed, having η and α approach 0 improves the algorithm's stability. Thus, we chose $\eta = 0.775$ and $\alpha = 0.45$. The chosen error threshold was 0.2.

We used 1,500 to 10,000 patterns for training, reserving the remaining 400 to 5,000 patterns for the generalization phase.

Table 1. Comparison of different implementations during the training phase. The number of epochs or iterations listed are those needed to achieve the RMS error shown.

Implementation	No. of training patterns	No. of epochs or iterations	Time (s)	RMS error	Patterns correctly classified (percentage) A	B	C	D
TMS320C80	1,500	200	1,806	0.20	85.2	87.1	80.6	80.9
PC with Matlab	2,000	3,000	683	0.24 (1.0)	90.0	98.9	66.1	82.1
PC with Matlab	10,000	3,000	3,231	0.16 (1.0)	91.0	98.0	66.0	84.0
Totem board	2,000	5,000	196	7,000 (20,000)	94.4	98.8	82.5	85.4
Totem board	10,000	5,000	929	7,000 (20,000)	96.9	98.7	80.7	88.2

Training results

We trained the ANN with different sets of patterns and stopped the learning process when the average squared error appeared to be flat, showing good classification capability. We found this flattening to be the primary indicator of acceptable classification performance, regardless of the value of the average squared error itself. This observation proves that a smaller average squared error does not always correspond to improved performance. Table 1 shows the percentage of points that the network correctly classified during the training phase, the execution times of the various implementations, and the corresponding root-mean-square (RMS) errors. The numbers in parentheses, under RMS error, represent the maximum range of values assumed by the output neurons. So, for example, an RMS error of 0.24 (absolute value) with respect to an output range of 1 corresponds to a probability of a bad network response of 24 percent. Likewise, if the RMS error is 7,000 with an output range of 20,000, the probability of a bad response is nearly 30 percent.

In porting the ANN application onto the TMS320C80 processor, we implemented a master/slave topology. The on-chip RISC processor acts as the master processor and triggers the training and generalization phases. The parallel processors are the slaves, sharing neurons and weights among themselves. The master processor synchronizes the activity of the four parallel processors. Using this topology required two modifications to improve performance:

- *Transformation of floating-point operations into integer ones.* Because the parallel processors don't have floating-point units, we transformed floating-point operations into integer ones. This step accounts for an 80 percent reduction in computation time.
- *Tabulation of the sigmoid activation function on prefixed points.* To obtain values of the function for nontabulated points, we tabulated the sigmoid activation function on prefixed points; the values of the function referring to nontabulated points could then be obtained through interpolation when requested. This reduced computation time by another 20 percent.

Speedup and efficiency

We calculated the speedup of the original program's parallelization by comparing the serial program's execution time on a single parallel processor with that of its parallel version executed on four parallel processors. The speedup was nearly 1.9, with a *parallelization efficiency*—the ratio between this effective speedup and the theoretical one represented by the number of processors (four, in this case)—of 47.5 percent. The overall execution times, however, were much longer than those achieved on the master processor. Converting the variables and tabulating the activation function provided a significant speedup, accelerating both the serial and parallel programs by a factor of 6.25. Nevertheless, the program's new parallel integer version achieved a parallelization efficiency close to 40 percent, which is reasonable because tabulating the activation function and transforming floating-point operations into integer ones diminished the parallel processors' overall computational load. This low efficiency is probably due to the communication overhead, which dominates the computation time because the application is small.

On the other hand, the limited on-chip

memory restricts the number of applications that can be directly implemented on the device. Thus, larger ANNs would require a wider addressable external memory, despite the accompanying decrease in performance. In any case, execution times are very long compared with those of the Totem implementation.

Table 2. Comparison of the recognition capability of Totem and PC with Matlab, during the generalization phase.

Architecture	No. of generalization patterns	Patterns correctly classified (percentage) A	B	C	D
PC with Matlab	3,000	91.1	98.2	63.8	84.8
Totem	700	91.3	95.2	87.3	81.9
Totem	3,000	93.9	96.5	77.9	84.6

Finally, we implemented the same problem on both the PC with Matlab tools and the Totem PCI board. We ran 3,000 epochs on the PC, whereas the Totem board took only 5,000 RTS iterations to reach the same RMS error. Although the number of RTS iterations was higher, our results show that Totem performed 5,000 RTS iterations in a shorter time (by a factor of about 3.5) than 3,000 BP epochs.

Generalization results

Table 2 presents the percentage of correctly classified points of the plane for the Matlab and Totem implementations. The data shows comparable performance for both implementations. Therefore, combining the results in Tables 1 and 2, we see that Totem achieved a similar level of correct classifications but with a 3.5-times shorter training phase.

The results from this first application point to the Totem PCI board with special-purpose processors as the most promising candidate for ANN processing. The Matlab implementation's performance was comparable, but the digital signal multiprocessor was clearly not the best choice, at least for problems of this size. For this reason, we focused only on the Totem board and Matlab implementations for the second application—a problem implying a heavier computational load.

Application 2: Generation of vestibular nystagmus

For the next application, we implemented a typical biomedical-data fitting problem: reproducing the generation of *vestibular nystagmus*, a reflexive eye movement important to scientists because it represents the standard approach for the clinical evaluation of a patient's vestibular system. Nystagmus is an oculomotor response comprising alternating slow and quick phases, which occur when the head undergoes sustained rotation, as well as during the sustained movements of a visual scene—for example, when a person looks through the window of a moving vehicle. Vestibular nystagmus is typically elicited by sinusoidal head rotation in darkness through a signal originating solely from the vestibular apparatus—a small structure in the inner ear's bony labyrinth that tells the brain how the head is moving. Thus, vestibular nystagmus is interesting because it occurs in the absence of optical stimuli, providing information on the functionality of vestibular organs.

Most sensorimotor functions develop at least partially as a result of adaptive and learning processes. These processes can occur during the development of an animal species or during a particular animal's lifetime. Researchers have demonstrated the role of such processes in the development of simple and well-studied neuromotor systems, such as the oculomotor system, the part of the central nervous system responsible for controlling eye movements. Researchers have also reported various examples of long- and short-term adaptive and learning behaviors—eye tracking of a target,[15] vestibular adaptation to artificially altered conditions of visual-vestibular interaction,[16] and so forth. Besides these well-known processes, other mechanisms in the oculomotor system show learning and adaptive abilities taking place both during the evolutionary development of the human species and during an individual's lifetime. One such learning behavior occurs during generation of vestibular nystagmus.

Explanation of vestibular nystagmus

Vestibular nystagmus consists of slow and quick phases. Slow phases, generated by the vestibulo-ocular reflex, compensate for head rotation. Fast phases countercompensate for the

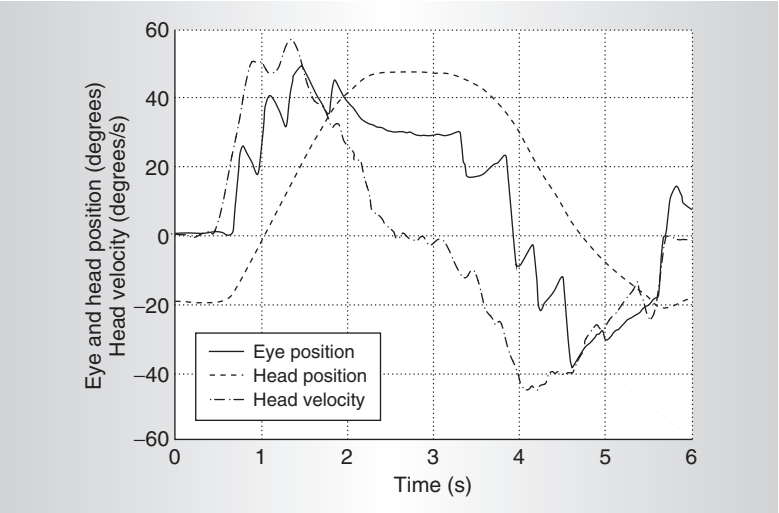

Figure 4. Reconstructed gaze signal.

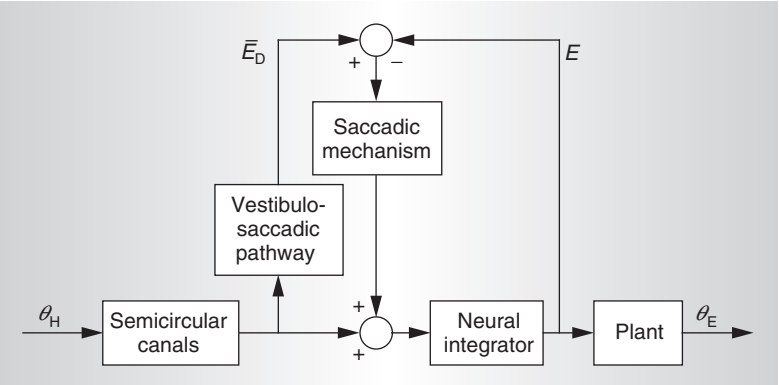

Figure 5. Model to reproduce eye-head coordination.

slow-phase movements and have saccadic origin. Saccades are the fastest eye movements that the oculomotor system produces. A person uses them to gaze quickly at an object. Active head rotations can evoke a similar pattern of eye movements in darkness, suggesting that vestibular nystagmus represents a specific strategy of eye-head coordination during natural behavior. Schmid and Zambarbieri proposed a mathematical model of vestibular nystagmus, postulating a direct "vestibulo-saccadic pathway (sequence of neural connections)" that conveys head rotation information from the peripheral vestibular system to the saccade-generating mechanism in the brain stem.[17] Ohki, Shimazu, and Suzuki found physiological evidence supporting the existence of such a pathway.[18]

When someone moves his head in darkness, his central nervous system activates an eye-head coordination strategy that yields vestibular nystagmus. This strategy exploits the head movement information to drive eye rotation so that the gaze anticipates and amplifies the head movement, thus allowing exploration of a greater portion of the scene.[19] You can appreciate such a strategy if you examine the traces of the gaze reconstructed by summing head position in space and eye position in the orbit, as Figure 4 shows. The gaze proceeds by alternating periods of fixation and rapid shifts. During the former, the central nervous system acquires visual information from the observed portion of the scene. During the latter, the visual axis corresponding to the saccadic component of vestibular nystagmus shifts rapidly. This strategy allows the subject to visually explore the dark surroundings to look for a possible object of interest.

We can reproduce this eye-head coordination strategy by introducing a model like that shown in Figure 5, into which we substitute an ANN for the vestibular saccadic pathway (VSP) block. This ANN consists of a two-layer perceptron—two input neurons, three nonlinear hidden neurons, and one output neuron. The VSP block receives both head angular velocity and displacement as input, and provides the desired eye position in the orbit as output.[20] The head velocity and displacement signals available to the VSP are those estimated by the peripheral vestibular system. Hence, the signals fed to the ANN correspond to the recorded head movement signals passed through a block modeling the dynamics of the semicircular canals (part of the vestibular apparatus). Within such a model, we trained the network to produce a continuous eye position signal, interpolating the eye position in the orbit at the end of each quick nystagmus phase. This eye position signal then goes to the saccadic mechanism.

Training setup

Although the training procedure adopted here is not physiologically likely, we can hypothesize a biological mechanism similar to that implemented by the ANN. In the model, we reconstructed the eye position signal from experimentally recorded signals and then fed to the network.[19] It is likely, instead, that the central nervous system progressively adjusts the synaptic weights in the VSP to pro-

duce a reference signal, responding to exploration efficiency criteria. These criteria could be either maximization of the explored scene or optimization of the number of fixations and their locations.

We implemented the ANN on both the PC with Matlab tools and the Totem board. The training phase involved 1,500 input patterns—with head position and velocity as input, and eye position as output—previously acquired in the same lighting conditions and from the same subjects.

Results

For the PC with Matlab, we ran 200 epochs, with an RMS error tolerance fixed at 5 percent between expected and real neuron output. This activation function evolves like the hyperbolic tangent, and Matlab functions automatically adjust the learning rate and the momentum constant.

In contrast, the Totem board required 2,000 RTS iterations to achieve nearly the same RMS error as the PC with Matlab. We chose the sigmoid curve as the activation function for this implementation. As Figure 6 shows, the model in Figure 5 works well, although its performance depends on the input patterns. Figure 7 reports the modeling error in the eye position (expressed in degrees). The Matlab and Totem outputs are from the generalization phase run for different test sets and after the same training. As Figure 7 shows, the errors are very close for the two architectures: 14.49° ± 3.16° for Matlab, 13.73° ± 2.79° for Totem. However, Totem performs better in 71 percent of the simulated trials.

The small differences in the two plots in Figure 7 could be due to contradictory patterns affected by biological noise that can drive the RTS algorithm off track more rapidly than the BP algorithm. Moreover, although the reported errors seem high, they pertain to the entire model and not only the ANN. In fact, if we consider only the outputs coming from the neural integrator, the difference between the two implementations is basically the same, but the absolute error decreases to about 5° or 6°.

Table 3 reports execution times on the different machines. The Totem-implemented ANN reaches the same RMS error 2.7 times faster than the Matlab implementation. Thus, even for this second application problem, Totem achieves similar levels of performance but with a far shorter training time. Totem's faster performance enables training the network using larger training sets, which should

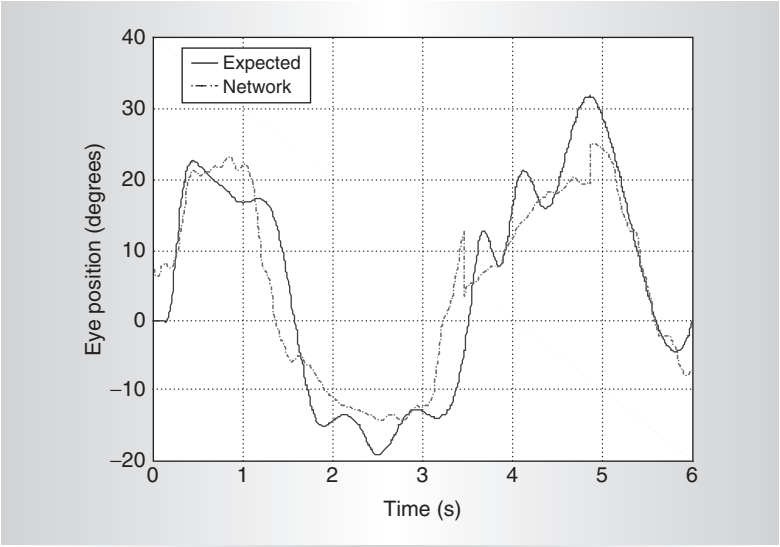

Figure 6. Expected and network outputs for different eye positions.

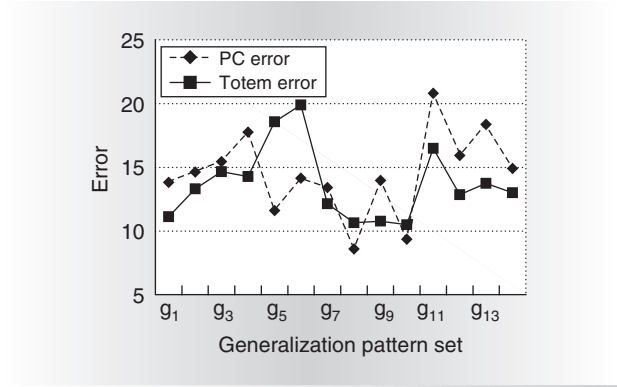

Figure 7. Error in determining eye positions, as measured by several generalization pattern sets.

Table 3. Execution times of the PC with Matlab and Totem implementations for the vestibular nystagmus problem.

No. of input patterns	Execution time (s)	
	PC with Matlab	Totem
7,500	400	150
1,500	85	31
360	33.5	12
300	27	10
225	21	8

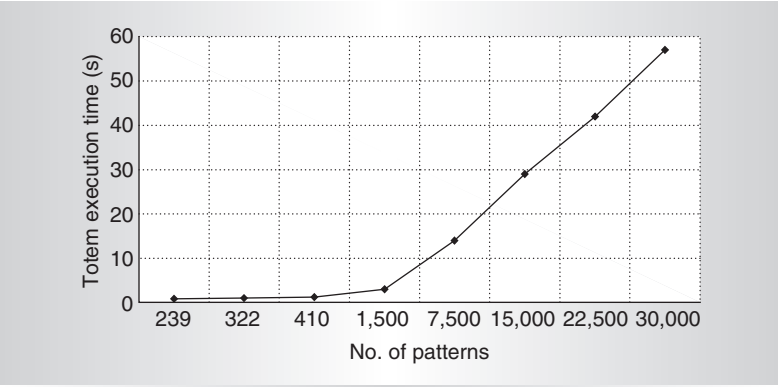

Figure 8. Variation of Totem execution times versus number of patterns, for a 500-iteration run on a 2 × 3 × 1 network.

improve generalization performance. As Figure 8 shows, execution times scale linearly with the number of patterns—a key characteristic, for instance, when facing large-scale problems. Hence, Totem should be able to preserve the speedup advantage it has over the other two implementations.

Of the three approaches tested, results indicate that Totem is the most promising. This board, composed of special-purpose processors, demonstrated good recognition capability along with excellent computing times in executing the RTS algorithm, which turns out to be more affordable than BP in the absolute minimum search of a cost function. Thus, the NC3001 chips in the Totem board are also suitable as a quick accelerator working with standard CPUs—particularly in industrial applications, where the processing power of the onboard microprocessor is not high, and an implementation must quickly evaluate complex models.

In our comparison, we didn't consider hand-coded ANN implementations. Indeed, we developed a C version of the first problem (2D classification), achieving a correct answer on average in 83 percent of the cases with execution times twice as long as those for the Totem board. This aspect, along with the long code development times, drove us to abandon the hand-coded ANN approach. Scientists usually don't like to lose time in codification details, but instead prefer to use ANN tools for investigation.

Because Totem performed about three times faster than the Intel 400-MHz processor, a battery of eight or 16 NC3001 chips could considerably improve computing power. This would allow parallel implementations of bigger networks that could handle more challenging problems. This is the path that NeuriCam is following with a few research groups. Together, they are implementing clusters of PCs, each equipped with one or more NC3001 chips on board, mainly for image-processing applications. And the NC3001's capabilities should increase as designers continually upgrade it to higher clock frequencies. In fact, a 60-MHz release is available that can further speed up the elaboration. The NC3001 also offers an easy-to-use Windows interface. This interface helps scientists avoid spending excessive time on detailed coding or error debugging, and speeds up neural network optimization. Finally, each Totem board currently costs about $1,000, including software. This price also makes it an economical choice for implementing ANN-based parallel systems. MICRO

Acknowledgment

We thank Giampietro Tecchiolli of NeuriCam for his useful advice and support throughout this work.

References

1. R. Yasdi, "A Literature Survey on Application of Neural Networks for Human Computer Interaction," *Neural Computing Application*, vol. 9, no. 4, Oct.-Dec. 2000, pp. 245-248.
2. S. Haykin, *Neural Networks: A Comprehensive Foundation*, Macmillan College Publishing, New York, 1994.
3. K. Narendra, "Neural Networks for Control: Theory and Practice," *Proc. IEEE*, vol. 84, no. 10, Oct. 1996, pp. 1385-1405.
4. A. d'Acierno et al., "A Parallel Implementation of the BP or Errors Learning Algorithm on a SIMD Parallel Computer," *Proc. Int'l Conf. Artificial Neural Networks* (ICANN 93), Springer-Verlag, New York, 1993, pp. 1074-1077.
5. E. Kerckhoffs et al., "Speeding up BP Training on a Hypercube Computer," *Neurocomputing*, vol. 4, nos. 1-2, 1992, pp. 43-63.
6. X. Liu et al., "Benchmarking of the CM 5 and the Cray Machines with a Very Large BP Neural Network," *Proc. Int'l Conf. Neural Networks* (ICNN 94), vol. 1, IEEE Press,

Piscataway, N.J., 1994, pp. 22-27.
7. N. Botros and M. Abdul-Aziz, "Hardware Implementation of an Artificial Neural Network Using Field Programmable Gate Arrays," *IEEE Trans. Industrial Electronics*, vol. 41, no. 6, Dec. 1994, pp. 665-667.
8. A. Petrowski et al., "Performance Analysis of a Pipelined BP Algorithm," *IEEE Trans. Neural Networks*, vol. 4, no. 6, Nov. 1993, pp. 970-981.
9. U.A. Muller et al., "Fast Neural Net Simulation with a DSP Processor Array," *IEEE Trans. Neural Networks*, vol. 6, no. 1, Jan. 1995, pp. 203-213.
10. D. Hammerstrom, "A VLSI Architecture for High-Performance, Low-Cost, On-Chip Learning," *Proc. Int'l Joint Conf. Neural Networks* (IJCNN 90), IEEE Press, Piscataway, N.J., 1990, pp. 537-542.
11. Y. Sato et al., "Development of a High-Performance, General Purpose Neuro-Computer Composed of 512 Digital Neurons," *Proc. Int'l Joint Conf. Neural Networks* (IJCNN 93), IEEE Press, Piscataway, N.J., 1993, pp. 1967-1970.
12. U. Ramacher et al., "Multiprocessor and Memory Architecture of the Neurocomputer SYNAPSE 1," *Proc. World Congress Neural Networks* (WCNN 93), vol. 4, Lawrence Erlbaum Associates, Mahwah, N.J., 1993, pp. 775-778.
13. S. Ma, C. Ji, and J. Farmer, "An Efficient EM Based Training Algorithm for Feedforward Neural Networks," *Neural Networks*, vol. 10, no. 2, Mar. 1997, pp. 243-256.
14. R. Battiti and G. Tecchiolli, "Training Neural Nets with the Reactive Tabu Search," *IEEE Trans. Neural Networks*, vol. 6, no. 5, Sept. 1995, pp. 1185-1200.
15. A.T. Bahill and J.D. McDonald, "Smooth Pursuit Eye Movements in Response to Predictable Target Motion," *Vision Research*, vol. 23, no. 12, 1983, pp. 1573-1583.
16. A. Gonshor and G. Melvill Jones, "Extreme Vestibulo-Ocular Adaptation by Prolonged Optical Reversal of Vision," *J. Physiology (London)*, vol. 256, no. 2, 1976, pp. 381-414.
17. R. Schmid and D. Zambarbieri, "Eye-Head Coordination during Active and Passive Head Rotations in Man," *Head-Neck Sensori-Motor System*, A. Berthoz, W. Graf, and P.P. Vidal, eds., Oxford Univ. Press, New York, 1991, pp. 434-438.
18. Y. Ohki, H. Shimazu, and I. Suzuki, "Excitatory Input to Burst Neurons from the Labyrinth and Its Mediating Pathway in the Cat: Location and Functional Characteristics of Burster-Driving Neurons," *Experimental Brain Research*, vol. 72, no. 3, 1988, pp. 457-472.
19. S. Ramat, *The Saccadic Mechanism Subserving Gaze Control*, doctoral dissertation, Biomedical Engineering, Polytechnic of Milan, 1999.
20. S. Ramat et al., "Learning Processes in the Vestibulo-Saccadic Pathway," *Proc. 1st Joint BMES/EMBS Conf.*, vol. 2, IEEE Press, Piscataway, N.J., 1999.

Giovanni Danese is full professor of computer science at the University of Pavia, where he heads the Microcomputer Laboratory. His research interests include computer architecture, computerized instrumentation, microprocessors, and parallel computing. Danese has a Laurea degree and a PhD in electronic engineering from the University of Pavia. He is a member of the IEEE Computer Society.

Francesco Leporati is an assistant professor of computer science at the University of Pavia. His research interests include microprocessors, systems on chips, computational physics, and ANNs. Leporati has a Laurea degree and a PhD in electronic engineering from the University of Pavia. He also had, in collaboration with INFM (the Italian Institute for the Physics of Matter), a postdoctoral fellowship for the design and analysis of parallel architectures dedicated to interacting particle systems. He is a member of the IEEE Computer Society.

Stefano Ramat is a contract professor of biomedical engineering at the University of Pavia. His research interests include motor control, mathematical modeling, neural networks, and signal processing. Ramat has a university degree in computer science engineering from the University of Pavia, a PhD in biomedical engineering from the Polytechnic of Milan, and a postdoctoral fellowship with the Ocular Motility Laboratory at Johns Hopkins University.

Direct questions and comments about this article to Francesco Leporati, Lab. Microcalcolatori, Dip. Informatica e Sistemistica, Univ. of Pavia, via Ferrata 1, 27100 Pavia, Italy; francesco.leporati@unipv.it.