The Ring Array Processor (RAP): Algorithms and Architecture

Nelson Morgan

International Computer Science Institute
1947 Center Street, Suite 600
Berkeley, CA 94704-1105, USA

BACKGROUND

In our speech recognition research, we have been experimenting with layered "neural" algorithms as probabilistic estimators for a Hidden Markov Model (HMM) procedure [1]-[3]. Features representing the spectral content of the speech are estimated 100 times per second. A layered network is trained to predict the phonetic label of each 10 msec "frame". This network takes as its input the spectral features from one or more frames, and has an output layer consisting of one unit per phonetic category. For some of our experiments, the input is continuous, and hidden layers are used. For others, the speech features are vector-quantized to map the frame into one of a set of prototype vectors, and the network input consists of a binary input neuron for each possible feature value, only one of which can be active at a time. In either case, the neural network is trained by back-propagation [4][5], augmented by a generalization-based stopping criterion [6]. It can be shown [7] that when the output targets are coded as 1 for the correct class and 0 for the others, and when a sum-of-squares error criterion is used, the trained output values are estimates of Maximum A Posteriori (MAP) probabilities, i.e. the probability of each class given the input features. This proof also is valid for some other common error criteria, such as relative entropy. When divided by the prior probabilities for each class, these values are proportional to the likelihoods which can be used as emission probabilities in the Viterbi decoding step of an HMM speech recognizer. A network can be useful for this procedure because it can estimate joint probabilities (joint over multiple features or time frames) without strong assumptions about the independence or parametric relation of the separate dimensions. We have conducted a number of experiments which seem to confirm the utility of this approach.

In the recognition process, computer resources are generally dominated by the primitives of dynamic programming (as used in the Viterbi decoding) - address calculations, reads, adds, compares, and branches (or conditional loads). This is particularly true for large vocabulary recognition. For example, for a 1000-word vocabulary we are using for recognition, a Sparcstation 1+ takes roughly 10 times real time to do the dynamic programming. The neural network calculations take about a second per second of speech (for the worst case of a large continuous input network). However, training via back-propagation is perhaps 5 times as long as the forward network calculation, and must be repeated over 10-20 iterations through a larger data set. Thus, the training runs we are currently doing can take from 12 hours to several days for a single speaker on the Sun. Planned experiments in feature selection will require a great deal more computing. Since our research is largely in the area of training algorithms as opposed to recognition per se,

a fast processor is required.

Extremely high performance speech processing systems can be built using special-purpose VLSI designs [8][9]. However, programmable systems with somewhat lower throughput can be designed using commercial general purpose microprocessors [10], and have the advantage of robust efficiency over a wider class of algorithms. For both approaches, custom system architectures can be used to streamline performance for a target class of algorithms[8]-[10]. Ring architectures have been shown to be a good match to a variety of signal processing problems [11] and neural network algorithms, including back-propagation [12][13][18]. The RAP design uses programmable floating-point digital signal processing (DSP) chips as the computational elements. We have connected the processors with a data distribution ring, implemented with programmable gate arrays. This programmability permits several variants of the basic ring operations, using a dedicated high-speed local communications channel. For a description of the hardware design and implementation, see [19].

ARCHITECTURAL CONSIDERATIONS

Artificial neural networks (ANNs) frequently do not have complete connectivity [14], even between layers of a feedforward network [15]. Nonetheless, an extremely useful subclass of these networks uses nonsparse connectivity between layers of "units", which are (for the most common case) nonlinear functions of the weighted sums of their inputs. The most common unit function uses a sigmoid nonlinearity, namely,

$$f(y) = \frac{1}{1 + e^{-y}} \qquad (1)$$

with

$$y = \sum_{i=1}^{N} w_i x_i + \theta \qquad (2)$$

where the w's are connection strengths (weights), the x's are unit inputs, θ is a unit bias, y is the unit potential, and $f(y)$ is the unit output.

The computational requirements of such algorithms are well matched to the capabilities of commercial DSPs. In particular, these circuits are designed with a high memory bandwidth and efficient implementation of the ‘‘multiply-accumulate’’ (MAC) operation (including addressing). However, if the unit implementations and corresponding weight storage are to be divided between multiple processors, there must be an efficient means for distributing unit outputs to all of the processors. If this is not provided in the system hardware, overall operation may be extremely inefficient despite efficient arithmetic. Full connectivity between processors is impractical even for a moderate number of nodes. A reasonable design for networks in which all processors need all unit outputs is a single broadcast bus. However, this design is not appropriate for other related algorithms such as the backward phase of the back-propagation learning algorithm, for which the weights are stored in the opposite order from the "forward" case described above.

More specifically, for a forward step the weight matrix should be stored in row-major form, i.e., each processor has access to a particular row vector of the weight

matrix. This corresponds to a list of connection strengths for inputs to a particular output unit. However, for a backward step the matrix should be distributed in column-major form, so that each processor has access to all connection strengths from a particular input unit. As Kung [9] has pointed out, the backward phase corresponds to a vector-matrix multiplication (as opposed to the matrix-vector multiplication of the forward case). One can use a circular pipeline or ring architecture to distribute partial sums to neighboring processors where local contributions to these sums can be added. Using this systolic mode of operation, partial sums for N units on N processors can be distributed in $O(N)$ cycles, where in contrast, a broadcast architecture would require $O(N^2)$ broadcasts to get all the partial sums to the processors where the complete sums can be computed.

Figure 1 shows the process of calculating the error terms for back propagation. The top table shows the initial location of the partial sums: s_{ij} refers to the ith partial sum (corresponding to the local contribution to the error term for hidden unit i) as computed in processor j. In other words, s_{ij} is all of the error term for hidden unit i which could be computed locally in processor j given the distribution of weights. In each step, each processor passes one partial sum to the processor on its right, and receives a partial sum from the processor on its left (with a ring connection between the end processors). The received sum is added into one of the partial sums. By choosing the passed values correctly, all processors can be usefully employed adding in values. Thus, in the example shown, each of the four processors have a completed error sum for a hidden unit after 3 steps. In general, $N-1$ steps are required to compute N such sums using N processors. Because of the ring hardware, the data movement operations are not a significant amount of the total computation, and multiple copies of the weights (each ordering of the weight matrix) or broadcasting individual weights are not necessary.

The forward calculations (as defined by equations 1 and 2) are speeded up by employing "read-shift" hardware to distribute layer outputs with minimal processor intervention. In this scheme, the processor signals the ring hardware to pass the data on to the next ring element by the act of reading the data from the ring. Thus, to "broadcast" data from all processors to all processors, each DSP writes to the ring once, and reads from it $N-1$ times. Including overhead, the cost would be

$$\#cycles = k \times (\ (\ (N-1) \times R) + W + S) + C \quad (3)$$

where N is the number of processors, R is the number of cycles per read, W is cycles per write, S is cycles for switching between read and write modes, and C is constant overhead for a loop which iterates k times. For the broadcast of 64 unit outputs, (a typical number for our application), and for the processor we have chosen (the Texas Instruments TMS320C30), this expression yields (for 16 nodes, the size of a typical system)

$$\#cycles = 4 \times (\ (\ (15) \times 1) + 2 + 2) + 8 = 84 \quad (4)$$

or 1.3 cycles per unit broadcast. The constant loop overhead can be minimized with in-line coding, where necessary. The major irreducible overhead in this total is due to the effects of the internal pipeline on the DSP chip, which causes delays for external writes and for mode switching between external reads and writes.

Initial Partial Sum Location			
P_1	P_2	P_3	P_4
s_{11}	s_{12}	s_{13}	s_{14}
s_{21}	s_{22}	s_{23}	s_{24}
s_{31}	s_{32}	s_{33}	s_{34}
s_{41}	s_{42}	s_{43}	s_{44}

Partial Sum Location After One Ring Shift			
P_1	P_2	P_3	P_4
s_{11}	s_{12}	$s_{12} + s_{13}$	s_{14}
s_{21}	s_{22}	s_{23}	$s_{23} + s_{24}$
$s_{34} + s_{31}$	s_{32}	s_{33}	s_{34}
s_{41}	$s_{41} + s_{42}$	s_{43}	s_{44}

Partial Sum Location After Two Ring Shifts			
P_1	P_2	P_3	P_4
s_{11}	s_{12}	$s_{12} + s_{13}$	$s_{12} + s_{13} + s_{14}$
$s_{23} + s_{24} + s_{21}$	s_{22}	s_{23}	$s_{23} + s_{24}$
$s_{34} + s_{31}$	$s_{34} + s_{31} + s_{32}$	s_{33}	s_{34}
s_{43}	$s_{41} + s_{42}$	$s_{41} + s_{42} + s_{43}$	s_{44}

Partial Sum Location After Three Ring Shifts			
P_1	P_2	P_3	P_4
$s_{12} + s_{13} + s_{14} + s_{11}$	s_{12}	$s_{12} + s_{13}$	$s_{12} + s_{13} + s_{14}$
$s_{23} + s_{24} + s_{21}$	$s_{23} + s_{24} + s_{21} + s_{22}$	s_{23}	$s_{23} + s_{24}$
$s_{34} + s_{31}$	$s_{34} + s_{31} + s_{32}$	$s_{34} + s_{31} + s_{32} + s_{33}$	s_{34}
s_{41}	$s_{41} + s_{42}$	$s_{41} + s_{42} + s_{43}$	$s_{41} + s_{42} + s_{43} + s_{44}$

Figure 1: Accumulation of partial error sums via the ring

For each board, the peak transfer rate between 4 nodes is 64 million words/sec (256 Mbytes/second). This is a reasonable balance to the 128 MFLOPS (64 million multiply-accumulates per second) peak performance of the computational elements. In general, units of a layer (actually, the activation calculations for the units) are split between the processors, and output activations are then distributed from all processors to all processors in the ring pseudo-broadcast described above. As long as the network is well-expressed as a series of matrix operations (as in the feedforward layered case), partitioning is done "automatically" when the user calls matrix routines which have been written for the multi-processor hardware.

STATUS

A six-layer printed circuit board was completed earlier this year, and was programmed to implement common matrix-vector library routines, and to do the forward and backward phases of back-propagation. Results of these tests are shown in Table I.

Table I: Single RAP Board Performance

MMACs = Millions of Multiply/Accumulates Per Second
MCPS = Millions of Connections Per Second
MCUPS = Millions of Connection Updates Per Second

operation	32x32 internal RAM	256x256 external static RAM
optimized matrix × vector	40.6 MMACS	61.4 MMACS
parameterized matrix × vector	29.1 MMACS	59.3 MMACS
parameterized forward propagation	23.5 MCPS	56.9 MCPS
parameterized forward plus learning	8.3 MCUPS	13.2 MCUPS

The first row of Table I shows the performance for a routine which has been hand-tuned for the specific matrix size. For a large enough dimension, this routine approaches 100% efficiency (with respect to 64 MMACS/board for a 16 MHz clock), and the single RAP board is roughly 50 times the speed of a Sun SparcStation 1 running the same benchmark. For smaller problems, such as the 32x32 case, the more generally useful "parameterized" routine (for which the dimension is a passed parameter) gives at most a 25% efficiency loss. Similarly, for the larger networks, the forward propagation performance becomes almost identical to the matrix-vector multiply (which is $O(N^2)$), since the sigmoid calculation (which is $O(N)$) becomes inconsequential in comparison to the multiply-accumulates. Finally, when learning is performed on each cycle (for a network with one hidden layer), the weight update steps dominate the computation. This is commonly the case with the back propagation algorithm, and similar ratios have been reported for other multi-processor implementations [13][16][17]. The update and the delta calculation each

require, on average, about as many arithmetic operations as the forward step, so that a factor of 3 decrease in throughput should be expected for the network calculation when learning is included. Another limitation is the DSP, which is optimized for dot-product calculations rather than the read-add-write of the weight update step. However, back-propagation performance on the Sun is similarly impacted by the costs of learning; learning on the Sun SparcStation takes about 3 times as long as forward propagation alone, and about 5 times as long as a simple matrix-vector multiply. In fact, the RAP appears to give roughly the same proportional speedup to user throughput both with and without learning. For an average of five floating-point operations per connection during learning, the last line of Table I corresponds to 41-66 MFLOPS, or roughly one-third to one-half of the peak arithmetic capability of the machine. For forward propagation, with an average of two floating point operations per connection, 89% of the peak arithmetic capability is obtained for the larger problem.

The board was then used for summer 1990 studies in feature extraction for continuous speech recognition [25][26]. With this single-board RAP we have already tackled computations that would have been impractical on a Sun workstation. We have now fabricated 3 boards, and are preparing to generate a second run with some minor modifications to permit RAP operation on Sun backplanes.

SYSTEM PERFORMANCE FACTORS

Extrapolation of performance measurements to a multi-board system is problem-dependent, but should be close to linear for layers of size $\geq 4N$. For the 32-bit host address space, the largest possible system would consist of 16 RAP boards (64 nodes), would have over 1 GB of memory, and would have a peak performance of 2 GFLOPS. Our simulations project a forward propagation performance in excess of 800 MCPS for such a hypothetical machine. Larger RAP systems could be built for a host with a larger address space. The size of the largest useful RAP is ultimately limited by the effect of parts of the application which cannot be partitioned between processors. For back-propagation learning in a layered feedforward network with one hidden layer, and an equal number of units in each layer, let

α_Q = number of cycles per connection (partitioned)

α_{L1} = number of cycles per unit (partitioned)

α_{L2} = number of cycles per unit (not partitioned)

C = constant overhead

N = number of units

P = number of processors

W = number of words of memory available for connection weights

then the total time in cycles for the algorithm would be

$$\alpha_Q \frac{N^2}{P} + \alpha_{L1} \frac{N}{P} + \alpha_{L2} N + C \quad (5)$$

For large values of N (where $\frac{N}{P}$ is held constant), we can ignore the second and fourth terms, so the machine would be 50% efficient or better when

$$\alpha_Q \frac{N^2}{P} \geq \alpha_{L2} N \qquad (6)$$

or

$$P \leq N \frac{\alpha_Q}{\alpha_{L2}} \qquad (7)$$

squaring and dividing by P,

$$P \leq \frac{N^2}{P} \frac{{\alpha_Q}^2}{{\alpha_{L2}}^2} \quad (8)$$

For the case of the maximum size weight matrix to fit in the static memory, this becomes

$$P \leq W \frac{{\alpha_Q}^2}{{\alpha_{L2}}^2} \quad (9)$$

From our test runs, we have estimated α_Q to be 9 for the back propagation calculation (7 when the weights are in internal memory) and α_{L2} to be 19, where the latter come from the delta calculation for the hidden layer, and from communication overhead. Therefore, an upper bound on processors for this problem appears to be roughly

$$P \leq W/4 \qquad (10)$$

Since our design includes 64K words of fast static RAM per node, the RAP would scale up to about 16000 processors (for sufficiently large back-propagation problems). At this point the code which was linear in the number of processors (such as communication costs, and the outer loop of the delta calculation for hidden units) would dominate the computation. Obviously, many practical problems such as synchronization would be much more difficult for a massively parallel version. Furthermore, the training time scales badly with large back-propagation nets, so that such a machine might be of no practical use.

Substituting the observed values for α_Q and α_{L2} into (8), we get roughly

$$\frac{N}{P} \geq 2 \quad (11)$$

This latter limit is of more immediate and practical concern. The RAP, used as a back-propagation machine, is reasonably efficient for networks with at least 2 units represented per processor. For small values of P (e.g., 4), a more detailed analysis (considering all four terms) suggests a preferred ratio of $\frac{N}{P} \geq 4$.

Estimating the constants from our measurements with the RAP, the machine cycles required to process one pattern (including learning on a P-processor RAP is

$$9\frac{N^2}{P} + 90\frac{N}{P} + 19N + 825 \qquad (12)$$

This is a good match to our measurements for larger values of N, and is conservative for smaller values, where we can make better use of memory. For most real examples, the contributions to the linear terms from communication are negligible. As mentioned earlier, the largest inefficiency in the system is the time required for weight updates, which typically dominates the α_Q term.

Using equation (12), we can infer the efficiency of parallelism for our target algorithm on a RAP of various sizes, as shown in Table II. This efficiency is the fraction of linear speed-up achieved withe P processors, for backpropagation on a layered network with N units in an input, single hidden, and output layer. Note that since the formula was empirically derived from RAP runs with one and four processors, only the last column is an extrapolation. For the problem sizes of interest to us in our speech work, for which layer sizes are 64 to 256, systems in the range of 4-16 processors (1-4 boards) give a respectable efficiency.

Table II: Efficiency of Parallelism
(Performance per processor relative to uniprocessor)

	#processors		
#units	1	4	16
16	1.00	.63	.25
32	1.00	.78	.42
64	1.00	.89	.62
128	1.00	.95	.78
256	1.00	.97	.88

The performance statistics given here must be compared with care to those reported for other machines, as there is no good suite of benchmarks or even common programs in use for this purpose. Some researchers have suggested using the Nettalk example, but we have not chosen to do this because of the ambiguity of the sparse input connectivity for that case - when a connection is not selected by an input, does that mean it has been updated by a value of 0.0? Some who have reported performance figures using this benchmark have treated these non-connections as updates. Others have used an auto-association example, which seems somewhat more reasonable. We chose to simply time the routines which are basic elements of any of our continuous input programs, and to implement back-propagation calculations for nets with layer sizes similar to those used in our actual application. The use of equal-sized layers simplified the empirical formulae derived above.

RAP APPLICATIONS

As described above, the primary target application for the RAP was back-propagation training of layered neural networks. However, we did not build special-purpose integrated circuits, which could have had a considerable speed advantage, because we wanted a programmable machine. While our current uses for the RAP continue to be dominated by back-propagation, we are able to modify the network algorithm daily for our experiments. Furthermore, we have experimented with using the RAP for computations such as the generation of Mandelbrot and Julia sets, and for dynamic programming. We also have used the RAP for calculation of dynamic features (first, second, and third temporal derivatives) to feed the layered network. While the topology has been optimized for the block matrix operations required in backpropagation, many algorithms can benefit from the fast computation and communication provided by the RAP.

In the case of dynamic programming, for instance, we currently treat a board as four independent processors that perform recognition on different sentences, thus speeding up a batch run. For real-time operation, the reference lexicon can be split up between processors, so that processors only need to communicate once for each speech frame. Thus, the RAP can be used as a SIMD machine (for our matrix operations, as in back-propagation), as a farm of separate SISD machines, requiring essentially no intercommunication (as in our current use for offline dynamic programming), or as a MIMD machine with simple and infrequent communication (as in the dynamic programming case for a distributed lexicon). Software is being developed to support all of these modes [20][21][22].

RELATED WORK

The simple communication ring topology used in the RAP is common to several other proposed and realized machines. Similar examples are the NeuroTurbo from Nagoya University [13] and the WARP [11][18]. The RAP differs from these most significantly in its use of Programmable Gate Arrays for communications hardware. These arrays permit easy modification of the low level register transfer logic to provide flexibility without sacrificing speed. In the iWARP (a commercial machine inspired by the WARP) a versatile communications processor performs data transfers between computing nodes with a latency of 100-150 nsec per word. The programmability of this communications processor favors the iWARP for systems using high-level complex protocols. The custom VLSI circuits used in the iWARP provide 60% of the computational capability of the TMS320C30 used in the RAP, with a significantly more sophisticated communications capability. The RAP, on the other hand, transfers words between DSP nodes within a single 62.5 nsec cycle using a very simple Programmable Gate Array design. In addition, the ability to customize this array for different low-level communication protocols without sacrificing performance is an advantage for the RAP. Back-propagation has been mapped to the WARP in a somewhat different way than what has been reported here [18]. Users of that machine chose to pass partial sums for the forward pass rather than the backward pass of our case; that is, they apparently stored the weights corresponding to the outputs of each unit, rather than the input. Another approach that they tried was to use each processor for a different copy of the complete network. Each copy operated on

different segments of the data, They ran multiple forward passes without updating the weights, and thus could read the weights in from a larger central memory. The resulting delay between forward passes and updates is probably acceptable for most applications, but it was not necessary on the RAP because of the large amount of fast local memory.

In contrast to both the RAP and the WARP, the NeuroTurbo uses dual-port memories between pairs of processors to implement the communication ring. The dual-port memory approach provides no hardware communication interlock but presents a simple model for user-designed software management of the ring. The designers apparently used a similar approach to the first one mentioned above, in which the communication of partial sums over the ring is done for the forward rather than the backward pass of the algorithm.

Two prominent examples of application-specific digital neural network systems that have been announced are the systolic Neuroemulator of Siemens [23], and the "X1" chip from Adaptive Solutions [24]. Although these will be comparatively fixed-function machines, they are expected to have significantly higher performance than the currently available alternatives, at least for the more common layered network algorithms.

SUMMARY

Ring architectures have been shown to be a good match to a variety of signal processing and connectionist algorithms. We have built a Ring Array Processor using commercial DSP chips for computational and Programmable Gate Arrays for communication. Measured performance for the first prototype board on target calculations is a factor of 20-50 higher than we have achieved on a general-purpose workstation. This tool is greatly aiding our connectionist research, allowing exploration of problems previously considered computationally impractical.

ACKNOWLEDGEMENTS

The RAP was a group development. James Beck [17], Jeff Bilmes [18], and Phil Kohn [19] provided the major sustained effort. Eric Allman and Joachim Beer were involved at early stages in the RAP design. Critical review was provided by Berkeley scientists and engineers too numerous to name, but certainly Jerry Feldman, Steve Omohundro, Jan Rabaey, and Joel Libove should be mentioned. Joel Libove also provided a more detailed hardware design review at critical stages. Herve Bourlard (formerly of Philips, now with L&H Speechproducts) provided the theoretical foundations for the speech application, and continues to collaborate with us on this work. Components were contributed by Toshiba America, Cypress Semiconductor, and Xilinx Inc., and Texas Instruments provided free emulators to debug the DSPs in-circuit. Finally, support from the International Computer Science Institute for this work is gratefully acknowledged.

REFERENCES

[1] H. Bourlard & N. Morgan, ‘‘Merging Multilayer Perceptrons and Hidden Markov Models: Some Experiments in Continuous Speech Recognition,’’ International Computer Science Institute TR-89-033, 1989.

[2] H. Bourlard, N. Morgan, & C.J. Wellekens, ‘‘Statistical Inference in Multilayer Perceptrons and Hidden Markov Models with Applications in Continuous Speech Recognition,’’ to appear in *Neuro Computing, Algorithms, Architectures and Applications*, NATO ASI Series, 1990.

[3] N. Morgan & H. Bourlard, ‘‘Continuous Speech Recognition Using Multilayer Perceptrons with Hidden Markov Models,’’ *Proc. IEEE Intl. Conf. on Acoustics, Speech, & Signal Processing*, pp. 413-416 Albuquerque, New Mexico, 1990.

[4] D.E. Rumelhart, G.E. Hinton &R.J. Williams, ‘‘Learning Internal Representations by Error Propagation,’’ in *Parallel Distributed Processing. Exploration of the Microstructure of Cognition.* vol. 1: Foundations, ed. D. E. Rumelhart & J. L. McClelland, MIT Press, 1986.

[5] P.J. Werbos,, ‘‘Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences,’’ Ph.D. thesis, Dept. of Applied Mathematics, Harvard University, 1974.

[6] N. Morgan, & H. Bourlard, ‘‘Generalization and Parameters Estimation in Feedforward Nets: Some Experiments,’’ International Computer Science Institute TR-89-017.

[7] H. Bourlard, & C.J. Wellekens, ‘‘Links Between Markov Models and Multilayer Perceptrons ,’’ in *Advances in Neural Information Processing Systems 1,* Morgan Kaufmann, pp.502-510, 1989.

[8] H. Murveit & R. W. Brodersen, ‘‘An Integrated-Circuit-Based Speech Recognition System,’’ IEEE Trans. Acoustics Speech and Signal Processing, Vol. ASSP-34, No 6, December, 1987.

[9] H. Murveit, J. Mankoski, J. Rabaey, R. Brodersen, T. Stoelzle, D. Chen, S. Narayanaswamy, R. Yu, P. Schrupp, R. Schwartz, & A. Santos, ‘‘A Large-Vocabulary Real-Time Continuous-Speech Recognition System,’’ *Proc. IEEE Intl. Conf. on Acoustics, Speech, & Signal Processing,* pp. 789-792, Glasgow, Scotland, 1989.

[10] R. Bisiani, T. Anantharaman, & L. Butcher, ‘‘BEAM: An Accelerator for Speech Recognition,’’ *Proc. IEEE Intl. Conf. on Acoustics, Speech, & Signal Processing*, pp. 782-784, Glasgow, Scotland, 1989.

[11] M. Annaratone, A. Arnould, H.T. Kung, & O. Menzilcioglu, ‘‘Using Warp as a Supercomputer in Signal Processing,’’ ICASSP ’86 Proceedings, Tokyo, pp. 2895-2898.

[12] S.Y. Kung & J.N. Hwang, ‘‘A Unified Systolic Architecture for Artificial Neural Networks,’’ *Journal of Parallel and Distributed Computing,* April, 1989, Michael Arbib ed.

[13] A. Iwata, Y. Yoshida, S. Matsuda, Y. Sato, & N. Suzumura, ‘‘An Artificial Neural Network Accelerator using General Purpose Floating Point Digital Signal Processors,’’ *Proceeding JCNN 1989*, pp. II-171-175.

[14] J.A. Feldman, M.A. Fanty, N. Goddard, & K. Lynne, ‘‘Computing with Structured Connectionist Networks,’’ in *Communications of the ACM, 1988.*

[15] Y. Le Cun, J. Denker, S. Solla, R. Howard, & L. Jackel, ‘‘Optimal Brain Damage’’ in *Advances in Neural Information Processing Systems II*- Morgan-Kaufmann, San Mateo, 1990 , David Touretzky, ed.

[16] S. Garth, ‘‘A Chipset for High Speed Simulation of Neural Network Systems,’’ First International Conference on Neural Networks, San Diego, pp. III-443-452, June 1987.

[17] X. Zhang, ‘‘An Efficient Implementation of the Back-propagation Algorithm on the Connection Machine CM-2,’’ in *Advances in Neural Information Processing Systems II*- Morgan-Kaufmann, San Mateo, 1990 , David Touretzky, ed.

[18] D. Pomerleau, G. Gusciora, D. Touretzky, & H. Kung, ‘‘Neural Network Simulation at Warp Speed: How We Got 17 Million Connections per Second,’’ in *IEEE International Conference on Neural Networks,* July, 1988, San Diego, CA.

[19] J. Beck, ‘‘The Ring Array Processor (RAP): Hardware,’’ International Computer Science Institute TR-90-048, 1990.

[20] J. Bilmes & P. Kohn, ‘‘The Ring Array Processor (RAP): Software Architecture,’’ International Computer Science Institute TR-90-050, 1990.

[21] P. Kohn & J. Bilmes, ‘‘The Ring Array Processor (RAP): Software Users Manual,’’ International Computer Science Institute TR-90-049, 1990.

[22] N. Morgan, J. Beck, P. Kohn, J. Bilmes, E. Allman, & J. Beer, ‘‘The RAP: a Ring Array Processor for Layered Network Calculations,’’ *Proc. of Intl. Conf. on Application Specific Array Processors,* pp. 296-308. IEEE Computer Society Press, Princeton, N.J., 1990.

[23] U. Ramacher and W. Raab, ‘‘Fine-grain System Architectures for Systolic Emulation of Neural Algorithms’’ *Proc. of Intl. Conf. on Application Specific Array Processors,* pp. 554-566. IEEE Computer Society Press, Princeton, N.J., 1990.

[24] D. Hammerstrom, ‘‘A VLSI Architecture for High-Performance, Low-Cost, On-chip Learning’’ *Proc. of IJCNN,* San Diego, June 1990.

[25] N. Morgan, C. Wooters, H. Bourlard, & M. Cohen, ‘‘Continuous Speech Recognition on the Resource Management Database using Connectionist Probability Estimation,’’ ICSI Technical report TR-090-044, also to be published in proceedings of ICSLP-90, Kobe, Japan.

[26] N. Morgan, H. Hermansky, C. Wooters, P. Kohn, & H. Bourlard, ‘‘Phonetically-based Speaker-Independent Continuous Speech Recognition Using PLP Analysis with Multilayer Perceptrons,’’ submitted to *IEEE Intl. Conf. on Acoustics, Speech, & Signal Processing,* Toronto, Canada, 1991.