Journal of VLSI Signal Processing, 6, 33–44 (1993)

Using Simulations of Reduced Precision Arithmetic to Design a Neuro-Microprocessor

KRSTE ASANOVIĆ AND NELSON MORGAN
International Computer Science Institute, 1947 Center Street, Berkeley, CA 94704
EECS Department, University of California at Berkeley, Berkeley, CA 94720

JOHN WAWRZYNEK
EECS Department, University of California at Berkeley, Berkeley, CA 94720

Received January 5, 1992; Revised June 2, 1992.

Abstract. This article describes some of our recent work in the development of computer architectures for efficient execution of artificial neural network algorithms. Our earlier system, the Ring Array Processor (RAP), was a multiprocessor based on commercial DSPs with a low-latency ring interconnection scheme. We have used the RAP to simulate variable precision arithmetic to guide us in the design of arithmetic units for high performance neurocomputers to be implemented with custom VLSI. The RAP system played a critical role in this study, enabling us to experiment with much larger networks than would otherwise be possible. Our study shows that back-propagation training algorithms only require moderate precision. Specifically, 16b weight values and 8b output values are sufficient to achieve training and classification results comparable to 32b floating point. Although these results were gathered for frame classification in continuous speech, we expect that they will extend to many other connectionist calculations. We have used these results as part of the design of a programmable single chip microprocessor, SPERT. The reduced precision arithmetic permits the use of multiple arithmetic units per processor. Also, reduced precision operands make more efficient use of valuable processor-memory bandwidth. For our moderate-precision fixed-point arithmetic applications, SPERT represents more than an order of magnitude reduction in cost over systems with equivalent performance that use commercial DSP chips.

1. Introduction

Our research group has developed several computer architectures for efficient execution of artificial neural network (ANN) algorithms. Our first system, the RAP [1], [2], was a multiprocessor based on commercial DSPs with a low-latency ring interconnection scheme. This system has been in serious use for over a year and has greatly accelerated our research into connectionist calculations, particularly speech recognition. Because the system was based on off-the-shelf components it was relatively easy to build, and the project took only one year from conception to working prototype. While this system continues to provide an important utility for our work, it is clear that a more efficient solution to ANN problems can be found by specializing our hardware further for the requirements of the target algorithms. In particular, the arithmetic precision can be reduced to that required for ANN algorithms.

While it as been commonly held that ANN calculations do not carry the high-precision requirement of mainstream scientific applications, few studies have verified this experimentally on any significant corpus of real-world data. We have used the RAP to simulate variable precision arithmetic to guide us in the design of higher performance neurocomputers based on custom VLSI. The computational power of the RAP system was necessary because our networks are significantly larger than those used in previous studies. The target application (continuous speech recognition) has typically used networks with from 20,000 to 300,000 connections, and are trained with large, real-world data sets of from 130,000 to 1,300,000 patterns. A ten board RAP system performing a large back-propagation task is almost three orders of magnitude faster than a Sun SPARCstation-2 running the same benchmark. Without the RAP this study could have taken years of computing time. Our study shows that, in contrast with scientific

applications, artificial neural network back-propagation training algorithms work well with moderate precision arithmetic. Results indicate that 16b weight values are sufficient to achieve training and classification results comparable to 32b floating point. Although our results were gathered for frame classification of continuous speech, we expect that they will extend to many other connectionist calculations.

Because of this reduced precision requirement, general purpose processors do not provide the most cost effective solutions for ANN problems. The performance of current VLSI processors is limited by both die area and interchip I/O constraints. Feasible die size is restricted by acceptable yields, and I/O bandwidth is restricted by available packaging technology. For current processor designs, die area limitations require that large memory systems must be implemented off-chip, making processor-memory bandwidth expensive. Excess precision results in inefficiencies in two significant ways. Firstly, the die area consumed by arithmetic units, in particular combinational multipliers, grows as the square of the word size. Secondly, excess precision operands waste valuable processor-memory bandwidth.

Therefore, we have developed a programmable single chip microprocessor, the Synthetic PERceptron Testbed (SPERT), to exploit reduced precision arithmetic. Reduced precision arithmetic results in smaller arithmetic units allowing multiple arithmetic units per processor, and makes more efficient use of processor-memory bandwidth. For our moderate-precision fixed-point arithmetic applications, SPERT represents over an order of magnitude reduction in cost over a system with similar performance based on commercial DSP chips.

In this article we first describe the experiments and hardware used to determine precision requirements and then outline the SPERT processor architecture designed to take advantage of these results. Section 2 describes the target speech frame classification problem and the neural network used in the precision requirements experiment. Section 3 describes the RAP machine, the system used in our experiments. Our experimental results are presented in Section 4. Our new processor design based on custom VLSI is presented in Section 5.

2. ANN Architecture

A phoneme-based speaker dependent continuous speech recognition system is under development. A layered Artificial Neural Network (ANN) is trained to classify 10 ms frames of speech as one of 60 to 70 phonetic classes. The speech recognizer uses the ANN to generate emission probabilities for a hidden Markov model (HMM) speech recognizer. Initial experiments indicate that this method compares favorably with conventional HMM speech recognition methods [3].

For some of our experiments, the network has 26 inputs. These are typically 12th order Perceptual Linear Prediction (PLP) coefficients [4] plus first order derivative terms for each speech frame. The input values are normalized to zero-mean and unity variance across the set of training data. For the simulations described here, these inputs are directly and fully connected to a layer of 256 hidden units. There are 61 outputs from the network, one per phone to be recognized. The hidden layer is directly and fully connected to the output layer. Note that while this is a large network (over 20,000 connections), it is actually quite a bit smaller than our best performing nets, which typically have over 100,000 connections. However, this net performs almost as well (65% in comparison to 70% correct frame classification), and was a more practical size to use for exploratory experimentation.

The database used consists of 500 sentences of continuous read speech from speaker DTD of the standard Resource Management RM-1 corpus, totaling 166023 10 ms frames. Each frame is annotated with a phonetic label derived from an iterative application of the Viterbi algorithm on a path constrained by a baseform transcription of the training set. The data is split into a training set of 400 sentences (131322 frames) and a test set of 100 sentences (34701 frames). The training set is used for the back-propagation learning algorithm. The test set is used to check the performance of the training algorithm to avoid over-fitting of the network to the training set.

There are two phases in the training scheme, which is a recent variant of the cross-validation learning approach used in [3]. Training starts by assigning some range of random values to all the weights, say $\pm r$. An initial value for α, the learning constant, is chosen and this value remains constant throughout the first training phase. A single training iteration consists of one pass through the training set, updating weights after every frame. After each training iteration, the net's performance is measured using the test set. When the performance improvement is less than some minimal improvement parameter $i\%$, the training scheme switches to the second phase. In the second phase of training, the value of α is halved before each iteration. This is essentially an annealing schedule for the stochastic

gradient search for a minimum in the error space. When the performance improves by less than i% during the second phase, training stops. The default values chosen for the training schedule parameters are initial random weights in the range $r = \pm 0.05$, initial $\alpha = 0.1$, and minimum performance improvement $i = 0.5\%$. Training performance has been found to be relatively insensitive to the exact value of these parameters.

3. The RAP System

The connectionist applications research in our group, particularly in speech recognition, requires extensive computational resources. The simulation of the effects of reduced precision arithmetic on these same connectionist applications requires even greater computational throughput. For these reasons, in 1989–1990, we designed and built a Ring Array Processor (RAP) for fast implementation of layered neural network algorithms [1], [2]. The project took roughly one year from conception to working prototype, and has since been in serious use for our applications research—we estimate that our group has performed computations that would have taken around a century on a SPARCstation-2. The RAP is a multi-DPS system (using Texas Instruments' floating point TMS320C30) with a significant amount of local memory (16 MB) per node, and a low latency ring interconnection scheme using programmable gate array technology. Multiple 4-processor boards serve as an array processor for a SPARC host. Theoretical peak performance is 128 MFLOPS/board, and peak ring communication speed is 64 MB/second/board. Test runs with up to ten boards (the largest configuration benchmarked) show a sustained throughput of roughly 10% to 90% of this for algorithms of current interest. For a sufficiently large network, forward propagation approaches 100% efficiency (with respect to 64 million multiply-accumulates/second/board for a 16 MHz clock), and ten RAP boards are two to three orders of magnitude faster than a Sun SPARCstation-2 running the same benchmark (typically 50–120 Million Connection Updates Per Second, or MCUPS, and 200–570 Million Connections Per Second, or MCPS). Table 1 shows a set of measured benchmark results on a 10-board system for a set of realistic network back-propagation problems using a single hidden layer.

A single broadcast bus is a reasonable design for a parallel system in which all processors need values from all other processors, e.g., for forward propagation in a fully connected layered network. However, it is difficult to design a high-performance bus that supports more than a handful of processors. A simple alternative is a ring design, which is also more flexible for other related algorithms such as the backward phase of back-propagation training. This and other issues of the RAP design are described in detail in [2].

Table 1. Measured performance, 10-board RAP.

Network Size	Forward Propagation	Full Learning
128 ⇒ 128 ⇒ 128	211 MCPS	63 MCUPS
512 ⇒ 512 ⇒ 512	527 MCPS	100 MCUPS
512 ⇒ 256 ⇒ 512	429 MCPS	83 MCUPS
234 ⇒ 1024 ⇒ 61	341 MCPS	45 MCUPS

The RAP runs in a VME-based system under the control of a SPARC 4/300 series host. An extensive set of software tools has been implemented with an emphasis on improving the efficiency of layered artificial neural network algorithms. This was done by providing a library of matrix-oriented assembly language routines, some of which use node-custom compilation. An object-oriented RAP interface in C++ is provided that allows programmers to incorporate the RAP as a computational server into their own UNIX applications. For those not wishing to program in C++, a command interpreter has been built that provides interactive and shell-script style manipulation of the RAP.

The RAP DSP software is built in three levels. At the lowest level are hand coded assembler routines for matrix, vector and ring operations. Many standard matrix and vector operations are currently supported as well as some operations specialized for efficient back-propagation. There is also a UNIX compatible library including standard input, output, math and string functions.

An intermediate level consists of matrix and vector object classes coded in C++. A programmer writing at this level or above can view the RAP as a conventional serial machine. These object classes divide the data and processing as evenly as possible among the available processing nodes, using the ring the redistribute data.

The top level of RAP software is the Connectionist Layered Object-oriented NEtwork Simulator (CLONES) environment for constructing ANNs [5]. The goals of the CLONES design are efficiency, flexibility and ease of use. Experimental researchers often generate either a proliferation of versions of the same basic program, or one giant program with a large number of options and many potential interactions and side-effects. By using an object-oriented design, we attempt to make

the most frequently changed parts of the program very small and well localized. The parts that rarely change are in a centralized library. Inheritance allows library classes to be easily modified to support experimental work.

4. Reduced Precision Simulations

We have used the RAP to simulate back-propagation calculations that we plan to execute on future VLSI hardware. Reduced precision weight representations were simulated by adapting an existing ANN training program, `mlp`, to call a weight quantization routine after each training pattern. The quantization routine rounds each updated floating point weight to the nearest value allowed by the simulated fixed-point representation, saturating if necessary. The effect of this is to simulate a processor that stores weights to some lesser precision between training patterns, but which performs all other arithmetic using the DSP's native 32-bit floating point precision. The TMS320C30's floating point format has a 24-bit significand, represented as a normalized two's-complement number with an implied most-significant nonsign bit. This allows the simulations to model two's-complement weights of up to 25 bits, where this number includes the sign bit.

The original `mlp` program used a table lookup with over 1000 entries to implement the sigmoid function. Reduced output precisions were simulated by quantizing the entries in this table.

The first series of experiments investigated the effect of changing the exponent of the fixed-point weights. The exponent of the MSB (sign bit) was varied in the range 0–5 for weight precisions of 12, 16, 20, and 25 bits. For example, an MSB exponent of 1 for a 16-bit weight would allow weights in the range ± 2. Biases were treated the same as weights. The default parameters listed above were used for the training schedule and output were kept at full precision. The network had 256 hidden units giving a total of 22589 weights and biases. The results are presented in figure 1. The graph plots the highest score achieved on the test set against the exponent of the MSB for each of the weight precisions.

The scores for 20b and 25b fixed point weights are comparable to those of the 32b floating point for MSB exponents in the range 1–5. The scores for the lower precision weights show much greater variation with MSB exponent. With 16b weights an MSB exponent of 3 provided the best scores, close to those of 32b floating point for MSB exponents in the range 1–5. The

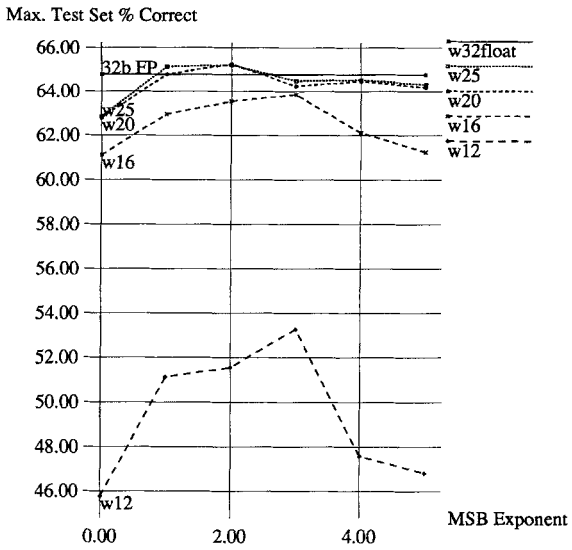


Fig. 1. Effect of varying exponent and precision. 256 hidden units, $\alpha = 0.01$, $r = \pm 0.05$, $i = 0.5\%$.

scores for the lower precision weights show much greater variation with MSB exponent. With 16b weights an MSB exponent of 3 provided the best scores, close to those of 32b floating point. The 12b weight runs also had the best performance with an MSB exponent of 3, though this gave over 10% lower scores than 32b floating-point.

The next series of experiments investigated a much larger number of weight precisions over a smaller range of MSB exponents. Weights in the range 12–20 bits and 25 bits were employed with MSB exponents in the range 1–3. The results are given in figure 2, plotted as best test score against number of weight bits for different MSB exponents. Overall, the scores are fairly insensitive to changes in the weight MSB exponent in the range 1–3, although lower precision weights (12–16b) have slightly better scores with a range of ± 8, while higher precision weights (19–25b) have slightly better scores with a range of ± 4. These results would seem to indicate that around 17–18 bits are required to achieve scores comparable to that of 32b floating-point, if the weights and biases are treated uniformly.

It was found that the higher scoring networks took roughly the same number of iterations (8–12) over the training set to converge as the 32b floating point version (9). Lower scoring networks tended to peter out in fewer training iterations (4–6). This indicates that there is no loss in training performance, as measured by number of training presentations required, for lower precision weights.

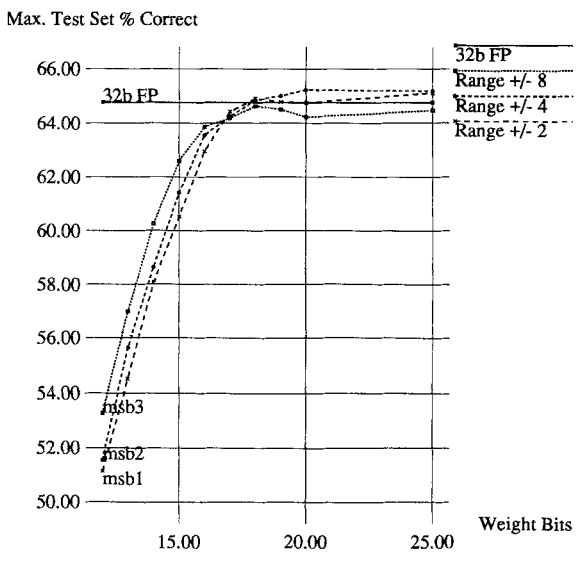


Fig. 2. Effect of varying exponent and precision. 256 hidden units, $\alpha = 0.01$, $r = \pm 0.05$, $i = 0.5\%$.

The weight and bias distributions resulting from these first experiments were examined. Figure 3 shows a bias histogram for the 32b floating point network after training, figure 4 shows the weight histogram. Figures 5 and 6 show bias and weight distributions for a 16b network with weights in the range ± 8.

These histograms show that weight values tend to be clustered around zero in a Gaussian distribution while bias values tend to be larger and negative. In the 16b net the bias values are seen to be hitting against

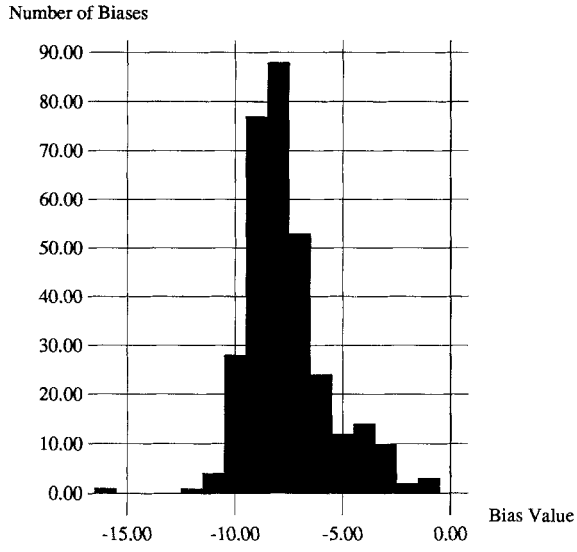


Fig. 3. Histogram of 32v float biases of trained network.

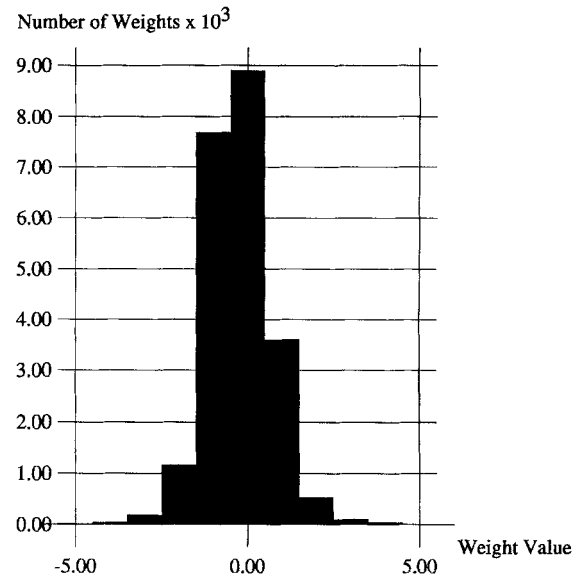


Fig. 4. Histogram of 32b float weights of trained network.

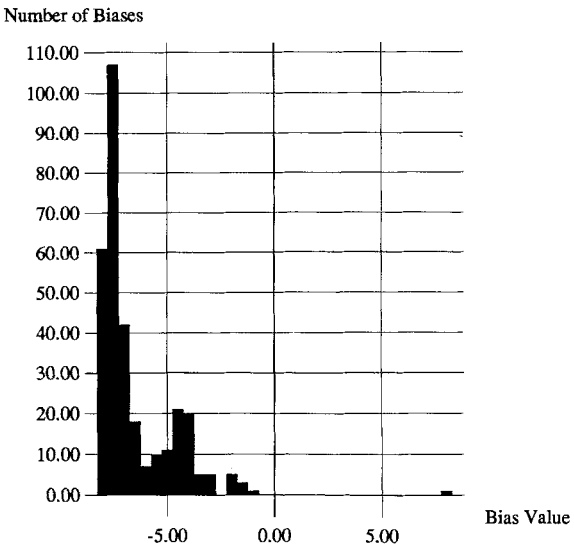


Fig. 5. Histogram of 16b biases of trained network. Biases rounded to range ± 8.

the range limitation. These histograms suggested that it would be beneficial to fix the bias exponent separately from the weight exponent.

In the next set of experiments, the bias exponent was varied over the range 2–5 while the weight exponent was set to 1–2. Figure 7 plots the results for weights in the range ± 2, and figure 8 plots the results for weights in the range ± 4. For short weight lengths, scaling the bias values separately gives a large improvement in scores. Higher weight precisions were fairly

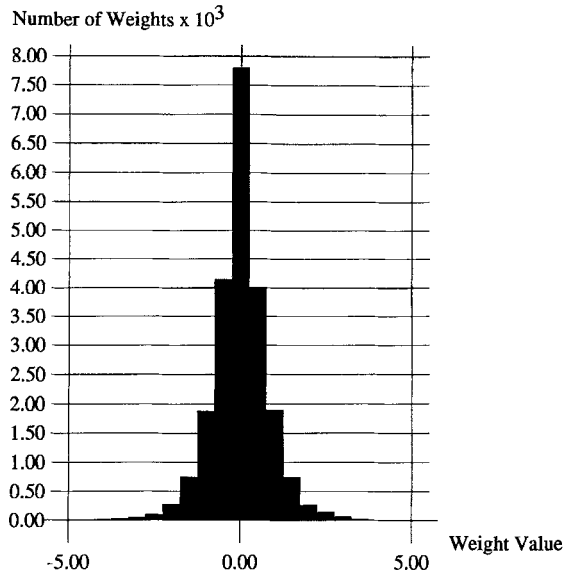


Fig. 6. Histogram of 16b weights of trained network. Biases rounded to range ±8.

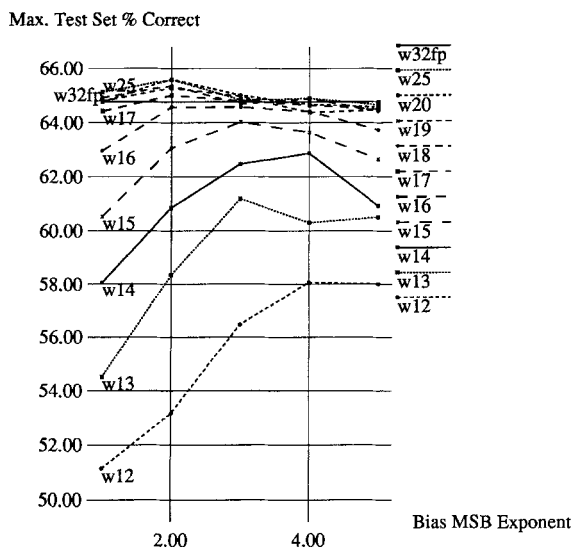


Fig. 7. Altering bias scaling. Weights in range ±2.

insensitive to changes in the bias exponent. These results show that 16b weights in the range ±2 with bias values in the range ±4–±16 give results comparable to 32b floating point.

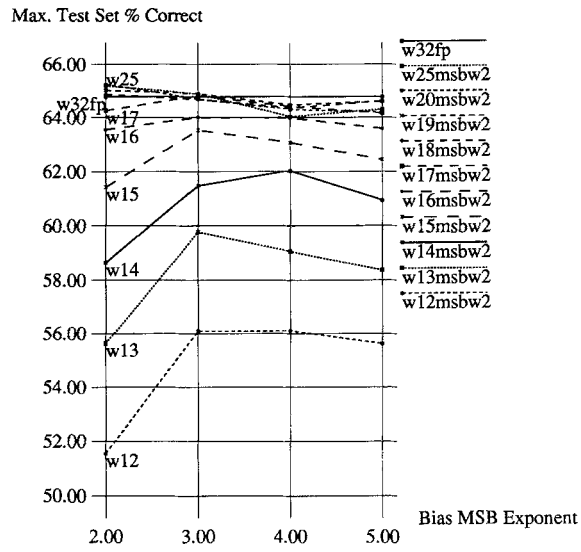


Fig. 8. Altering bias scaling. Weights in range ±4.

The results so far have used full precision for the output values. A series of simulations were performed reducing the precision of the output values. Figure 9 shows results for a network with 25b weights of range ±2 with biases of range ±4, and for a network with 16b weights of range ±2 with biases of range ±8. These scalings gave results for each weight precision in the previous experiments. Output precisions of 1, 2, 4, 8, and 16 and 25 bits were measured.

The results show that 16b weights and biases with 8b outputs give essentially the same scores as using

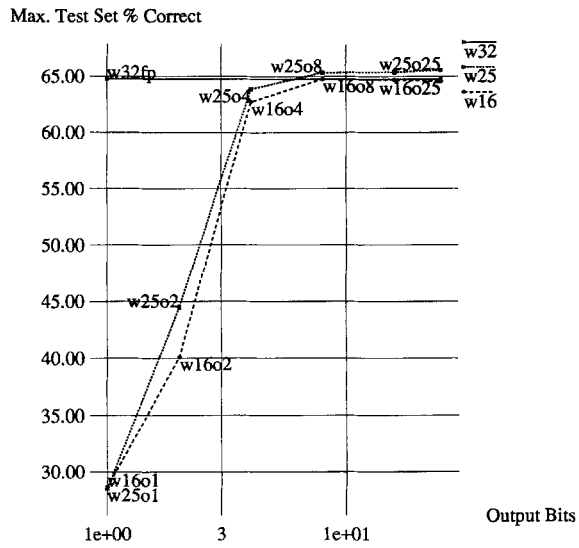


Fig. 9. Varying output precision. 25b and 16b weight networks.

32b floating point values throughout. Using less than 4 bits for the output precision gives noticeably poorer performance for networks with both 16b an 25b weights.

Initial experiments revealed that truncating intermediary weight values to the reduced precision produced much poorer results than using rounding. In figure 10, truncation and rounding results are plottted for various weight precisions. All weights and biases were rounded in the range ± 2 and outputs were kept at full precision. Note that there are more lower precision data points for the rounding curve, and more higher precision data points for the truncation curve. It can be seen that use of truncation reduces peformance by an amount corresponding to roughly 6 bits of precision when compared to rounding. This demonstrates the sensitivity of the back-propagation algorithm to the quality of the arithmetic employed. Examination of the trained weight histograms revealed that the truncated weight networks were in a very different region of weight space than the floating point network. This can be explained by noting that truncation adds an overall negative bias to the gradient descent causing the network to follow a very different learning trajectory, reaching a poorer local minimum.

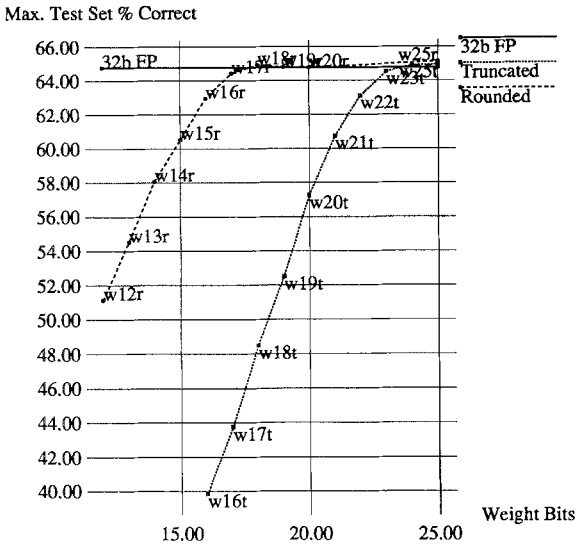


Fig. 10.. Truncation versus rounding.

Our results are in agreement with other researchers' findings with smaller networks and training sets in different application areas [6], [7], and with theoretical studies [8]. These results suggest an architecture optimized 16b × 8b multiply-accumulate arithmetic. A preliminary description of such a design follows below.

5. SPERT

We are developing a single chip CMOS microprocessor that will incorporate these reduced precision results. The architecture is fully programmable, and executes a wide range of connectionist computations efficiently. Special emphasis is being placed on support for variants of the commonly used back-propagation training algorithm for multi-layer feedforward networks [9], [10], as we have used for the speech classification task described earlier. At the design clock rate of 50 MHz, detailed performance analysis indicates that this chip will sustain over 300 MCPS when performing classification, and around 100 MCUPS when performing back-propagation training.

The Synthetic PERceptron Testbed (SPERT) fulfills an intermediate role in our neural network system development plans. The RAP gave us an early speed-up over existing workstations so that we could develop our speech algorithms using large databases (over 1,000,000 patterns) and large networks (100,000 to 400,000 connections). As described earlier, the use of commercial chips for this system permitted a short development time. SPERT is our first multiprocessor machine (tentatively named CNS-1), which will use similar technology. A consistent set of software abstractions, based on CLONES, will assist in porting code between these platforms.

This section provides an overview of the SPERT project. A more complete description can be found in [11].

5.1. SPERT Architecture

Based on the reduced precision experiments, our requirements for the arithmetic units in SPERT were that they provide fast 16b × 8b fixed-point multiplication, with efficient handling of larger (l4–32b) intermediary results.

We chose a uniform register and datapath width of 32b for SPERT. Adders, shifters, and limiters are relatively inexpensive to implement and all work on full 32b data to retain the precision of intermediary results. Adopting a uniform on-chip data width both simplifies the programmer's model and the implementation. Memory loads and stores can transfer vectors of lower precision operands making efficient use of processor-memory bandwidth. All memory operands of less than 32b are sign-extended to 32b when loaded.

We save datapath area by adopting a reduced precision multiplier. To simplify the implementation and

reduce routing area we required that the multiplier fit within a 32b datapath. After some experimentation, we discovered that our most compact 16b × 8b multiplier layout compatible with a 32b datapath could be readily extended to a 24b × 8b array with minimal cost in area and cycle time. Although the multiplier we use is somewhat larger than that suggested by the study, the memory interface has been optimized for 16b weights and allows one 16b operand to be fetched per cycle per datapath. Applications requiring higher precision may be memory bandwidth limited unless memory operands can be reused. The multiplier produces a full 32b result, but can optionally round the result to 24b, 16b, or 8b. The 8b input to the multiplier can be treated as either signed or unsigned to support higher precision multiplies in software; a 16b × 16b multiply occupies the multiplier for 2 cycles, a 24b × 24b multiply occupies the multiplier for 3 cycles. Thus, performance should degrade gracefully for applications requiring greater precision.

The overall structure of SPERT is shown in figure 11. The main components are an instruction fetch unit with an instruction cache, a scalar 32b integer datapath and register set, a SIMD array containing multiple 32b fixed point datapaths each with an associated register file, a 128b wide external memory interface, and a JTAG interface and control unit. (Note: JTAG are the initials of the Joint Test Action Group who helped formulate the IEEE1149.1 boundary scan standard. For brevity, the term JTAG is used to refer to the standard in this paper).

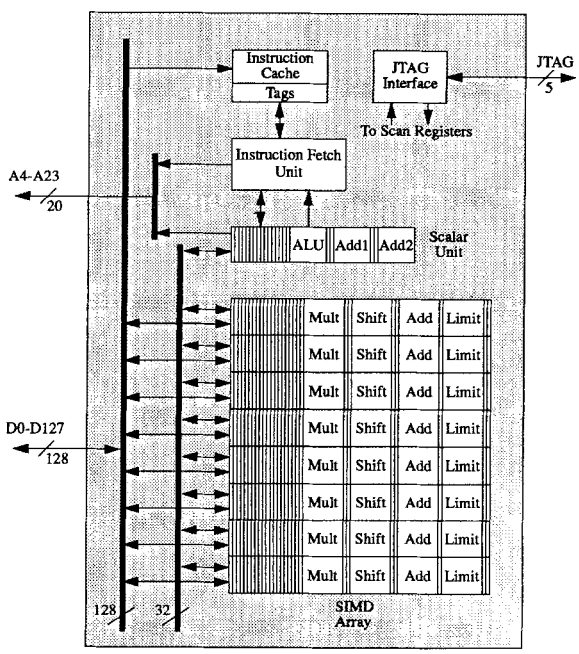


Fig. 11. SPERT Structure.

The initial implementation of SPERT has a maximum clock frequency of 50 MHz, and all performance figures in this document are based on this clock rate. SPERT systems may run at slower clock rates to accommodate larger memory systems.

SPERT is intended to function as an attached processor for a conventional workstation host. An industry standard JTAG serial bus is used as the host-SPERT interface. Chip and board testing, bootstrap, data I/O, synchronization, and program debugging are all performed over the JTAG link. Data I/O over JTAG can be overlapped with SPERT computation.

SPERT is a VLIW machine, with a single 128b instruction format. The instruction pipeline has 7 stages, and, in the absence of instruction cache misses and host memory accesses, one instruction can be completed every cycle. The greatly expanded instruction bandwidth requirement can be a problem for highly parallel VLIW architectures. However, typical SPERT applications are dominated by small inner loops, and a relatively small on-chip instruction cache should give excellent performance.

The scalar unit is similar to the integer datapath of a RISC processor, and is used for general scalar computation and to support the SIMD array by providing address generation and loop control. It contains a triple-ported 16×32b general purpose register file, a fully bypassed ALU, and two 32b address generators `add1` and `add2`.

The SIMD array contains 8 fixed-point datapaths, similar to those found in DSP chips. Each SIMD datapath contains a 24b × 8b multiplier, a 32b saturating adder, a 32b shifter, and a 32b limiter. The limiters allow a 32b word to be clipped to an 8b, 16b, or 24b value, saturating the output if necessary. All of these functional units can be active simultaneously, delivering up to 400 × 10^6 fixed-point multiply-accumulates per second. Each SIMD datapath includes a triple-ported 16 × 32b general purpose register file. In addition each SIMD datapath contains a number of datapath registers. These are located on the inputs of the functional units, and allow results to be moved between functional units without requiring extra read and write ports on the general purpose register file. Each SIMD datapath has 3 global buses, each of which can transfer two values per clock cycle, together with several local sneak paths between physically adjacent functional units. This rich interconnect coupled with the separate

input registers, supplies the high operand bandwidth required by the highly parallel datapaths.

The external memory interface supports up to 16 MB of single cycle memory over a 128b data bus. At the maximum 50 MHz clock rate, 12ns access time SRAMs are required and memory bandwidth is then 8900 BM/s. The relatively small external memory will typically be used to hold weight values and program code. Input and output training patterns will be supplied by the host from a large database held on disk. Even with relatively small networks, the time spent training on one pattern is sufficient to transfer training data for the next pattern over the JTAG link.

5.2. *SPERT VLSI Implementation*

SPERT is being implemented in a 1.2 μm CMOS process using MOSIS scaled CMOS design rules, with a target 50 MHz clock rate. The design uses a mixture of full-custom cells for the datapaths and pads, and standard cells for control and other random logic. The current floorplan indicates a die size of approximately 7.4 $\times$ 9.2 mm^2.

A number of scaled SPERT components have been successfully fabricated and tested on 2.0 μm MOSIS TinyChips. These include the triple ported register file, a saturating adder, and a JTAG controller with custom boundary scan cells. The multiplier has been designed and simulated thoroughly and is out for fabrication. The custom cells we are developing are also being made available as Lager library cells. A full description of the VLSI cell library work is available in [12].

5.3. *SPERT Performance Analysis*

During the architectural design process, a number of applications were considered and used to evaluate design alternatives. The primary envisaged application is back-propagation training and in this section we present detailed performance results for back-propagation training on SPERT.

These results have been obtained by careful hand analysis of assembly code routines. The results include all loop overhead, blocking overhead, and instruction cache miss cycles. The routines are fully parameterized, with all parameters passed in scalar registers. For these timings, all input data resides in memory, and all output data is placed back into memory in a form that will allow these routines to be chained with no penalty for data rearrangement. The only restriction imposed by these routines is that the number of units in a layer be a multiple of 8. Usually this is not an important restriction, but dummy units can be inserted to pad smaller layers if necessary. For these results, weights are 16b, inputs are 8b, and activations are 8b sigmoids looked up in a 64K entry table. Most intermediary values are calculated to 24–32b precision.

Figure 12 plots the performance of SPERT on forward propagation. The graph plots a number of curves for a fully connected pair of layers, varying the number of units in the input and output layers. Peak performance is around 352 MCPS. There are nine instructions in the inner loop for forward propagation. The first instruction brings in a vector of 8 $\times$ 8b inputs; the subsequent 8 instructions load in 8 vectors of 8 $\times$ 16b weights each. Each of the weight vectors will be multiplied by one of the 8 inputs, and the 8 results are accumulated one per datapath. By using reduced precision, SPERT makes better use of processor-memory bandwidth, and can sustain 89% of its theoretical peak performance even when all operands are stored in external memory. The fast on-chip scalar units helps SPERT attain high performance even with smaller nets. Nets must be smaller than 32 $\times$ 32 to drop below half peak performance.

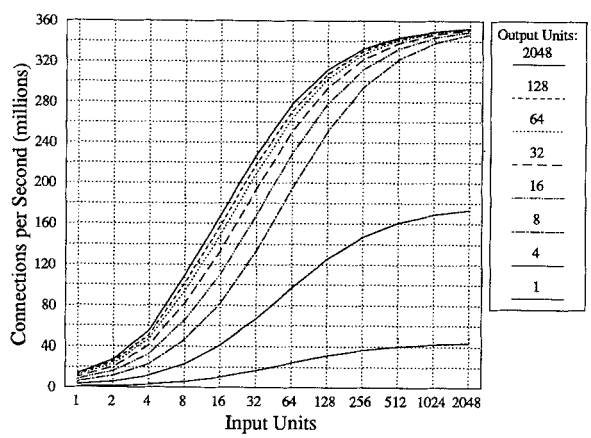


Fig. 12. SPERT forward propagation performance.

Figure 13 shows the training performance on a 3 layer feedforward network with equal numbers of units on the input, hidden, and output layers. Peak performance is around 100 MCUPS.

In comparing these performance figures with other implementations, it must be stressed that the figures for SPERT represent training with no pooling of weight updates. The training is entirely on-line with weight

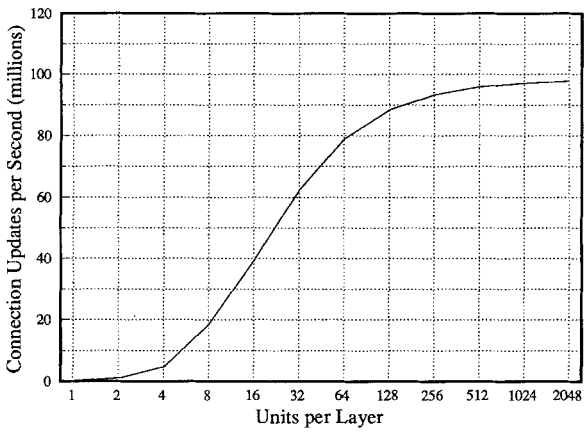


Fig. 13. SPERT training performance. Three layers, with equal number of units per layer.

updates occurring after every pattern presentation. Some parallel systems train multiple copies of a network, then periodically pool the weight updates. This can lead to reduced training performance, counteracting the benefits of increased parallelism.

5.4. Related Work

Several related efforts are underway to construct programmable digital neurocomputers, most notably the CNAPS chip from Adaptive Solutions [13] and the MA-16 chip from Siemens [14].

Adaptive Solutions provides a SIMD array with 64 processing elements per chip, in a system with four chips on a board controlled by a common microcode sequencer. As with SPERT, processing elements are similar to general purpose DSPs with reduced precision multipliers. Unlike SPERT, this chip provides on-chip SRAM sufficient to hold 128K 16b weights but has no support for off-chip memory. Larger networks require additional processor chips, even if the extra processing power cannot be efficiently employed.

Like SPERT, the MA-16 leverages the high density and low cost of commercial memory parts. This chip is a direct realization of three general network formulae that are intended to summarize many connectionist computations. The system that is envisioned will consist of a 2D systolic array containing 256 of these chips, the resulting system will provide impressive raw peak throughput. However, the purely systolic approach, together with deep pipelines and relative inflexibility will severely limit its general applicability. In particular, the systolic array will perform poorly on smaller networks, and on related code that is not purely connectionist in nature.

SPERT will provide the large off-chip memory bandwidth of the Siemens chip along with the general programmability of the Adaptive Solutions chip. In addition, SPERT provides high scalar performance and performs well on smaller layers. These capabilities are important for our application, as we are interested in experimenting with larger, sparser network structures that are organized as collections of smaller, highly interconnected, sub-networks. Both other architectures will experience a serious drop in performance if network size is reduced, or if network structure is made more complex. Both the CNAPS and the MA-16 are designed to be cascaded into larger SIMD processor arrays. An important goal in the SPERT design is to prototype ideas for a parallel processing node that will be used in a future, scalable, MIMD multiprocessor system. Such a system would target large, irregular network structures.

6. Conclusion

We have described a simulation study in which reduced precision arithmetic was used for back-propagation training of an ANN in a large speech classification application. This study was made possible by the availability of the RAP, a fast neurocomputer designed previously in our group. The study showed that 16-bit weights and 8-bit unit outputs gave sufficient performance of our target application, provided that weights and biases are scaled separately and that round-to-nearest was used to reduce provision of intermediate values. These findings were used to select the arithmetic units for the preliminary design of a VLSI single-chip microprocessor specialized for these applications. The microprocessor design also permits the use of higher precision arithmetic with a gradual degradation in performance, so that we retain significant capabilities for tasks outside of our current experience. All major VLSI blocks of the design have currently either been successfully fabricated, or have at least been simulated at the circuit level. The full system will be implemented in 1992.

Acknowledgments

This report is part of a related chain of projects involving contributors from both the Computer Science

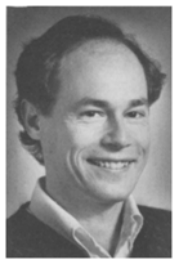
Division at Berkeley and the Realization Group at ICSI. Brian Kingsbury and Bertrand Irissou did most of the detailed VLSI design for the elements of SPERT. James Beck designed the RAP hardware and is a collaborator in the SPERT design. Phil Kohn, who did much of the RAP software, provided software support for our reduced precision simulations. The National Science Foundation provided support for the VLSI building blocks and design tools through Grant No. MIP-8922354, and also with Graduate Fellowship support for Brian Kingsbury. John Wawrzynek received support from the National Science Foundation through the Presidential Young Investigator (PYI) award MIP-8958568. The larger project continues to be supported by the International Computer Science Institute.

References

1. N. Morgan, J. Beck, E. Allman, and J. Beer, "RAP: A ring array processor for multilayer perceptron applications," *Proc. IEEE Intl. Conf. on Acoustics, Speech, & Signal Processing*, 1990, pp. 1005–1008.
2. N. Morgan, J. Beck, P. Kohn, and J. Bilmes, "Neurocomputing on the RAP," In K.W. Przytula and V.K. Prasanna, eds, *Digital Parallel Implementations of Neural Networks*, Englewood Cliffs, NJ: Prentice Hall, 1993, In Press.
3. N. Morgan and H. Bourlard, "Continuous speech recognition using multilayer perceptrons with hidden Markov models," *Proc. IEEE Intl. Conf. on Acoustics, Speech, & Processing*, 1990, pp. 413–416.
4. H. Hermansky, "Perceptual linear predictive (PLP) analysis of speech," *J. Acoust. Soc. Am.*, vol. 87, April 1990.
5. P. Kohn, "CLONES: Connectionist Layered Object-oriented Network Simulator," Technical Report TR-81-073, International Computer Science Institute, 1991.
6. T. Baker and D. Hammerstrom, "Modifications to artificial neural network models for digital hardware implementation," Technical Report CS/E 88-035, Department of Computer Science and Engineering, Oregon Graduate Center, 1988.
7. M. Duranton and J.A. Sirat, "Learning on VLSI: A general purpose digital neurochip," *Proc. IJCNN-89*, pp. II-613, Washington D.C., June 1989.
8. J.L. Holt and J.N. Hwang, "Finite precision error analysis of neural network hardware implementations," *Proc. IJCNN-91*, 1991, pp. I-591–525.
9. D.E. Rumelhart, G.E. Hinton, and R.J. Williams, "Learning internal representations by error propagation," *Parallel Distributed Processing. Exploration of the Microstructure of Cognition*, Vol 1, Cambridge: MIT Press, 1986.
10. P.J. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*, Ph.D. Dissertation, Dept. of Applied Mathematics, Harvard University, 1974.
11. K. Asanović, J. Beck, B.E.D. Kingsbury, P. Kohn, N. Morgan, and J. Wawrzynek, "SPERT: A VLIW/SIMD microprocessor for artificial neural network computations," Technical Report TR-81-0972, International Computer Science Institute, 1991.
12. B.E.D. Kingsbury, K. Asanović, J. Wawrzynek, B. Irissou, and N. Morgan, "Recent work in VLSI elements for digital implementations of artificial neural networks," Technical Report TR-91-074, International Computer Science Institute, 1991.
13. D. Hammerstrom, "A VLSI architecture for high-performance, low-clost, on-chip learning," *Proc. International Joint Conference on Neural Networks*, 1990, pp. II-537–543.
14. U. Ramacher, J. Beichter, W. Raab, J. Anlauf, N. Bruls, M. Hachmann, and M. Wesseling, "design of a 1st generation neurocomputer," in *VLSI Design of Neural Networks*, Boston: Kluwer Academic, 1991.

Krste Asanović received his B.A. degree in the Electrical and Information Sciences Tripos from Cambridge University in 1987. From 1983 through 1989 he was employed at the GEC Hirst Research Center, London, and worked on several projects including the design of a large scale associative processor for the ESPRIT project PADMAVATI. He is currently working towards his Ph.D. in Computer Science at the University of California, Berkeley. His research interests are in VLSI, massively powerful computer architectures, and object oriented programming languages.

Nelson Morgan is a Research Scientist at the International Computer Science Institute in Berkeley. He is also affiliated with the EECS Department at UC Berkeley, where he teaches from time to time. In 1980 he received his Ph.D. from this Department, where he worked with Robert Brodersen. Dr. Morgan has designed practical signal processing and pattern recognition systems based on connectionist algorithms for the last ten years. He also worked on the extraction of information about cognitive processing through the analysis of EEG signals. He has worked with speech and auditory processing problems for twenty years, was formerly in charge of speech research at National Semiconductor, and currently conducts research into connectionist recognition of continuous speech. He was also the principal architect for the parallel neurocomputer, the Ring Array Processor (RAP).



John Wawrzynek is an Assistant Professor of Electrical Engineering and Computer Sciences at the University of California, Berkeley, where he teaches courses in VLSI system design. He is a recent recipient of the National Science Foundations Presidential Young Investigator award. John received his Ph.D. from the California Institute of Technology where he worked under Carver Mead, and holds a Masters of Science in Electrical Engineering from the Unviersity of Illinois, Urbana/Champaign. He was an original member of Silicon Compiler Systems Corp. (formally Silicon Compilers Inc.), and has consulted for several companies including Schumberger Palo Research Lab. and USC Information Sciences Institute (MOSIS project). John's current research interests include custom VLSI for parallel computation, computer aided design of integrated circuits, and signal processing.