SPERT: A VLIW/SIMD Microprocessor for Artificial Neural Network Computations

Krste Asanović [*] [†]
James Beck [*]
Brian E. D. Kingsbury [*] [†]
Phil Kohn [*]
Nelson Morgan [*] [†]
John Wawrzynek [†]

TR-91-072
January 1992

Abstract

SPERT (Synthetic PERceptron Testbed) is a fully programmable single chip microprocessor designed for efficient execution of artificial neural network algorithms. The first implementation will be in a 1.2 μm CMOS technology with a 50MHz clock rate, and a prototype system is being designed to occupy a double SBus slot within a Sun Sparcstation.

SPERT will sustain over 300×10^6 connections per second during pattern classification, and around 100×10^6 connection updates per second while running the popular error backpropagation training algorithm. This represents a speedup of around two orders of magnitude over a Sparcstation-2 for algorithms of interest. An earlier system produced by our group, the Ring Array Processor (RAP), used commercial DSP chips. Compared with a RAP multiprocessor of similar performance, SPERT represents over an order of magnitude reduction in cost for problems where fixed-point arithmetic is satisfactory.

[*] International Computer Science Institute, 1947 Center Street, Berkeley, CA 94704
[†] EECS Department, University of California at Berkeley, Berkeley, CA 94720

1 Introduction

We are developing a single chip CMOS microprocessor that will provide significant improvements in cost/performance for artificial neural network calculations. 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 backpropagation training algorithm for multi-layer feedforward networks[RHW86, Wer74]. This class of networks is of interest in the speech recognition task that is the focus of our applications work [MB90]. In this work, networks are trained to estimate phonetic probabilities for use in a Hidden Markov Model based continuous speech recognizer. At the design clock rate of 50 MHz, detailed performance analysis indicate that this chip will sustain over 300×10^6 connections per second (CPS) when performing classification, and around 100×10^6 connection updates per second (CUPS) when performing backpropagation training.

In earlier work, our group designed a programmable machine called the Ring Array Processor (RAP) for general backpropagation training of layered feedforward networks [MBAB90, MBKB92]. This used multiple TMS320C30 DSP chips [Tex88] connected in a ring array. SPERT's performance is comparable to that of a 40 node RAP multiprocessor. Such a RAP system occupies ten 9U VME boards, while in contrast the initial SPERT system design will be a double SBus card that will fit into a workstation. SPERT achieves this dramatic improvement in cost/performance through a combination of using reduced precision fixed point arithmetic, providing high on- and off-chip operand and instruction bandwidth, and employing a highly parallel architecture. SPERT has multiple parallel datapaths each with parallel and pipelined functional units that can execute up to 37 primitive operations per cycle (1.8 GOPS).

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). The use of commercial chips for this system permitted a short (one year) development time. SPERT is our first full microprocessor, and is the result of several years of work on our VLSI design path. SPERT tests this path as a first step towards our next full multiprocessor machine (tentatively named CNS-1), which will use similar technology. A consistent set of software abstractions will assist in porting code between these platforms.

This paper provides an overview of the SPERT project. Section 2 summarises work to determine arithmetic precision requirements for neural network algorithms, and the resulting selection of arithmetic units for SPERT. Section 3 presents the overall architecture of SPERT. Section 4 describes the VLSI implementation methodology and current status. The SBus board design is covered in Section 5, and Section 6 outlines the planned software environment. Section 7 contains results of a detailed performance analysis of SPERT for the backpropagation training task. Section 8 compares SPERT to other programmable neurocomputers.

2 Reduced Precision Arithmetic

One common feature of connectionist algorithms that can be exploited when developing a special purpose neurocomputer is that arithmetic operations can be performed with much less precision and dynamic range than is required for typical scientific and engineering codes. Shorter word widths require smaller datapaths. In particular, the area of a fast multiplier circuit grows as the square of the width of the input operands. Shorter word widths also require less operand storage and consume less memory bandwidth per operand. Employing fixed point rather than floating point arithmetic also significantly reduces design complexity and functional unit latencies.

To guide the development of the arithmetic units in SPERT, we have conducted a series of experiments to investigate the effects of moderate precision, fixed point arithmetic on the backpropagation training algorithms used in the speech recognition application. The RAP was used to simulate reduced precision arithmetic while training large networks on large, real world datasets [AM91].

Connection weights dominate storage and memory bandwidth requirements, so we first investigated the effects of reducing only weight precision. The tests were performed by adapting an existing program to call a weight quantization routine after each training pattern. The effect of this is to simulate a special purpose processor that stores and updates weights in fixed point to some lesser precision, but which performs all other arithmetic using 32b floating point. Initial results suggest that 16b weight values are suf-

ficient to give essentially the same training and classification performance as full 32b floating point values, but that special care must be paid to scaling and rounding. In particular, simple truncation of wider range weights to 16 bits seriously hurt performance for our learning task (typically reducing phonetic discrimination from a 65% accuracy to less than 40%), while unbiased rounding yielded essentially no degradation compared to the floating point weights. A second set of experiments varied unit output precision with 25b and 16b weights, and demonstrated that 4–8 bits of output precision were sufficient for training and classification. These results are consistent with other studies in a number of different application areas [BH88].

Based on these 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 (24–32b) 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.

To simplify the implementation and reduce routing area, we required that the multiplier fit within a 32b datapath. After some experimentation, we discovered that the 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 that 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. By increasing the multiplier latency to two clock cycles, we both simplified the design, and reduced its area, without reducing throughput. A new multiply can be initiated every cycle, but cannot use results from the immediately previous instruction. Envisaged applications have sufficient parallelism to hide this latency. 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.

With a compact multiplier design, multiple fast datapaths can be integrated on a single die. The main compute engine in SPERT is a small SIMD array of 8 parallel datapaths, with each 32b datapath containing a 24b×8b multiplier, a 32b adder, a 32b arithmetic/logical shifter, and a 32b limiter. The shifter when shifting left can optionally shift in 1/2 LSB to prepare an operand for a rounding addition. The limiter allows a 32b word to be clipped to an 8b, 16b, or 24b value, saturating the output if necessary.

3 SPERT Architecture

The overall structure of SPERT is shown in Figure 1. The main components are a JTAG[1] interface and control unit, 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, and a 128b wide external memory interface.

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 accomodate large 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. The host calls routines on SPERT by initializing the instruction pointer over the JTAG link. The host performs data I/O by reading and writing SPERT external memory over JTAG. A single memory transfer moves 128b of data between the memory and the data shift register, stalling SPERT for one cycle. The shifting of the data register can be overlapped with further SPERT memory accesses. The JTAG interface is clocked at 50 MHz providing a maximum host–SPERT data bandwidth of 6 MB/s. A `synchflag` bit in the SPERT control unit supports host–SPERT synchronization. Each SPERT instruction has a `synch` bit which when set

[1] 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 document.

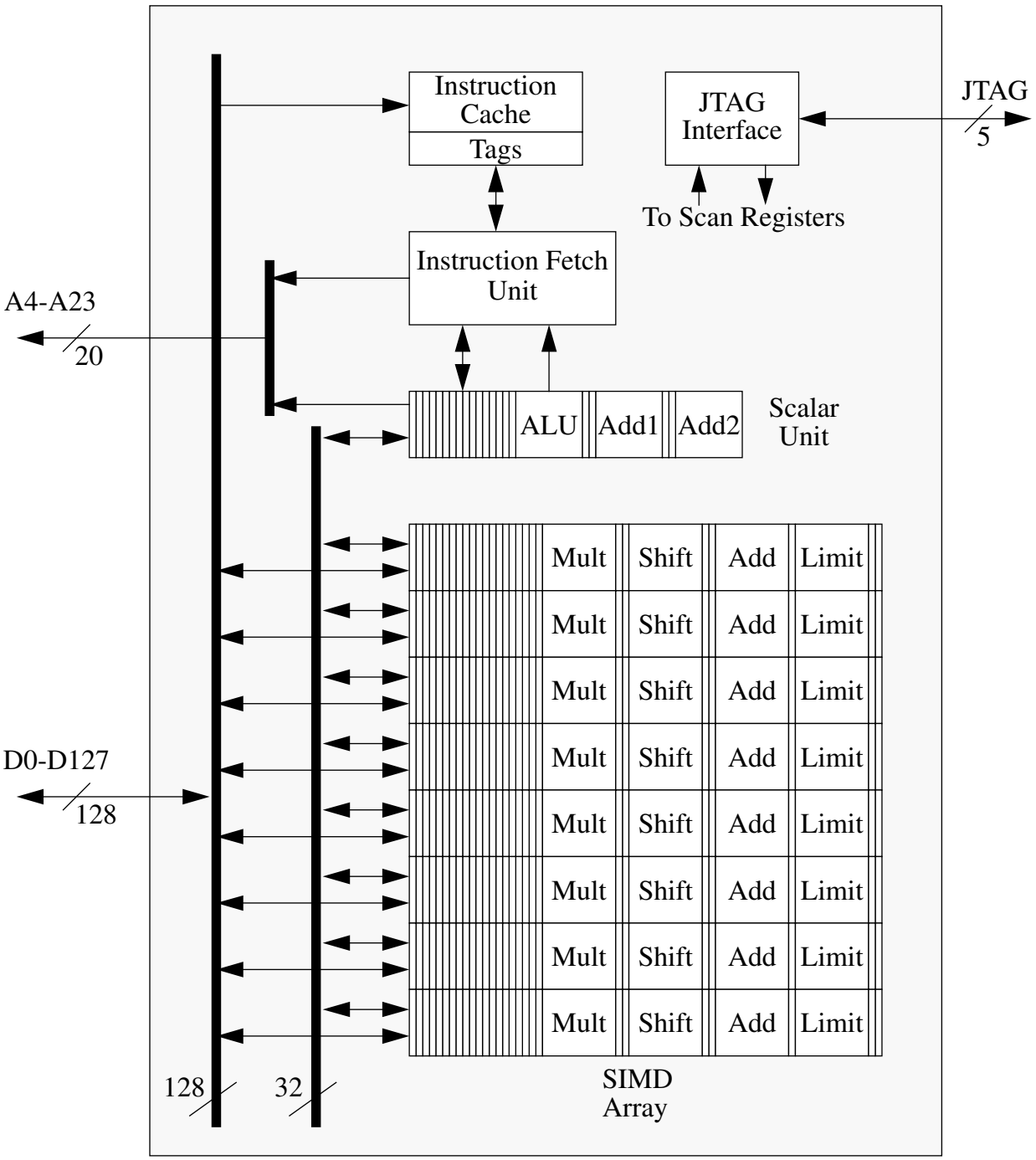

Figure 1: SPERT Structure.

causes SPERT to complete this instruction, set the `synchflag` bit and enter an idle state. The host can poll the value of `synchflag` over JTAG, detect when it becomes set, perhaps initiate some I/O task, then clear `synchflag` to let SPERT proceeed at the next instruction. This facility is also used to allow single step execution of SPERT programs for debugging purposes; if all instructions have `synch` set, then SPERT will pause and wait for the host after every instruction. These additional JTAG functions have been carefully designed so as not to compromise JTAG's use as a test port.

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 gives excellent performance. The instruction cache holds 16 instructions and is direct mapped with a separate tag per cached instruction. The instruction cache on SPERT has been organized as a *stall* cache [ASPF92] to take advantage of the single cycle external memory. During instruction fetch, the stall cache bypasses the instruction cache whenever the instruction currently in the memory access stage of the instruction pipeline does not use the memory port, and fetches the instruction direct from memory. The 128b memory bus allows a complete 128b VLIW instruction to be fetched each cycle, and so cache misses only cost one stall cycle each. Given the performance of the small stall cache and the large external memory bandwidth, there is little incentive to use a denser instruction encoding for SPERT, and hence we avoid extra decode complexity and delay.

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 two address generators each have two separate 32b address registers to allow them to operate independently of the main scalar ALU. The instruction fields for the scalar unit are more horizontally encoded than for a typical RISC unit, allowing better utilization of the processor components in highly parallel code. For example, the scalar register file can use one read port to supply a register to which the immediate will be added to form a vector load address, while the second read port is used to obtain a value to be broadcast to the SIMD array, while the write port is used to update the register file with the result of a scalar load from a SIMD register. Here, the three register file ports are used individually for three unrelated parallel operations.

The scalar unit includes a logic unit, and this is connected to the branch comparator. The branch comparator determines whether or not to branch depending on the sign and equality to zero of the result of the logic unit. This allows branches dependent on the result of a logical operation to complete in a single cycle. The two instructions following a branch are always executed, regardless of the branch outcome.

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. 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 800 MB/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.

SPERT is a pure load/store architecture; all memory transfers are to/from registers, and all functional unit operations are register-register. A memory access may transfer either a single 8b, 16b, or 32b scalar operand, or a vector of 8×8b, 8×16b, or 4×32b operands. All SPERT registers are 32b wide, and all memory operands less than 32b wide are sign-

extended to 32b when loaded.

A scalar memory load operand can be loaded into the scalar unit and/or broadcast to all 8 datapaths. A vector load can only move into a SIMD register. The 4×32b vector load updates only 4 of the 8 SIMD datapaths. Similarly for a 4×32b store only one half of the datapaths participate. All vector memory accesses move a contiguous, aligned vector of operands and the assignment of datapaths to vector elements is fixed. These restrictions dramatically simplify the memory interface. If more complex memory access patterns are required, then these can be performed through the scalar unit.

The scalar unit experiences a two cycle load delay. The SIMD unit is active later in the pipeline than the scalar unit, and experiences no load delay. However, there is a single cycle store delay; the value being stored cannot have been produced in the preceding cycle.

The scalar unit can directly load/store into a SIMD register, treating each one as a small 8×32b on-chip memory. There are two main uses for this capability. The first is to allow 8 scalar operands to be transferred between registers and external memory in one cycle. The other use is to enable a tight coupling between the scalar unit and the SIMD units for cases where operations cannot be parallelized across the SIMD datapaths. SPERT is capable of performing a scalar unit load from a SIMD register, a scalar unit store to a SIMD register, and an external memory SIMD vector access in a single instruction.

4 SPERT VLSI Implementation

A major decision in any VLSI design is the choice of clocking strategy. Our VLSI group has been experimenting with the "True Single Phase Clocking" (TSPC) proposed in [YS89] for CMOS circuits, gaining experience with the technique for larger and more complex systems [Waw92]. These designs result in reduced complexity, higher density, and higher speeds than conventional CMOS clocking methodologies. In TSPC designs, there is a single clock signal distributed to two complementary latch types; one is transparent on clock low, the other on clock high. To control skew and simplify timing analysis, we classify chip signals as either clock or data and forbid gating the clock signal with data signals. The result is that there is only a single clock signal providing the timing reference, and data signals only have to meet setup and hold times relative to the clock edges. In the SPERT design, a single central clock buffer will be used. This helps ensure low clock skew by eliminating delay variations introduced by intermediate clock distribution buffers.

SPERT is being implemented in a 1.2 μm CMOS process using MOSIS scaled CMOS design rules, with a target 50MHz 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 $7.4 \times 9.2\,\mathrm{mm}^2$. Our tool set is a mixture of commercial and public domain software, the latter coming primarily from the various UC Berkeley toolsets. We are using `magic` as our primary layout editor, SPICE3 and CAzM as our circuit level simulators, `irsim` as our switch level simulator, and the Lager/ViewLogic system for schematic capture and logic synthesis.

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 and saturating adder. A JTAG controller with custom boundary scan cells has also been fabricated and delivered, and has passed initial tests. The multiplier design has been completed and is out for fabrication. The custom cells we are developing are being made available as Lager library cells. A full description of the VLSI cell library work is available in [KAW+91].

5 SPERT SBus Board

We are designing an SBus board around the SPERT chip. The board will contain a SPERT chip clocked at 33MHz, 2 MB of 20ns SRAM, and the SBus to JTAG interface. This will be placed inside a SPARC desktop workstation to form a powerful, low cost neural network accelerator. This combination will form the core of a demonstration training system that we plan to implement in mid 1992. We intend to develop higher speed and larger memory SPERT systems as faster and denser memory parts become available.

The board will measure approximately 5"×6" with the layout shown in Figure 2. It will occupy two SBus slots and will dissipate less than 20W, with most of the power dissipation due to the SRAM. The SPERT chip will be packaged in a 209 pin PGA and will be socketed on the board. The same board will be used to test SPERT chips after fabrication.

The external memory consists of sixteen 128K×8b 20ns SRAMs. Four SSI bipolar address drivers are used to buffer the address line outputs from the SPERT chip. Each data bus pin of the SPERT chip has only a single connection to an SRAM data pin. This minimizes the load on the SPERT and SRAM

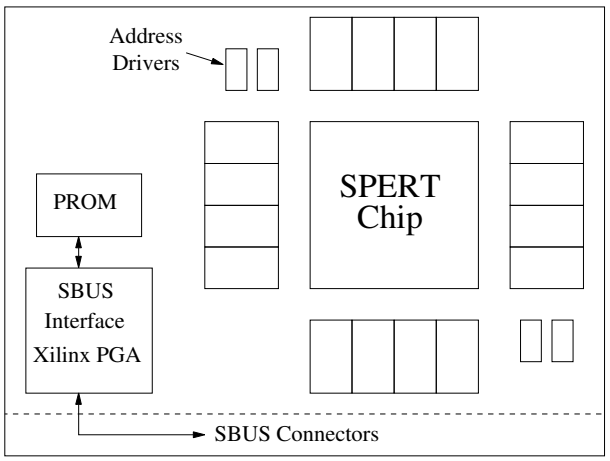

Figure 2: SPERT board.

data bus drivers, and simplifies board layout. To further reduce data bus load, and also to reduce board size, 32-pin surface-mount small outline packages will be used for the SRAMs.

The SBus interface will be implemented using a Xilinx Field Programmable Gate Array (FPGA) and a PROM. The FPGA converts the parallel data from the host to the serial format required by the JTAG standard used in the SPERT chip. The function of the PROM is to supply initialization data to the host CPU, as required of all SBus boards.

6 Software Environment

The application developers' interface to SPERT will be a set of C++ library classes for the host Sparcstation that implement a selection of common neural network related algorithms. We preserve our investment in software by using the same higher level simulator software that we have developed for the RAP. The matrix and vector libraries define a common interface that hide the SPARC, RAP or SPERT hardware from the user and the simulator code. The Connectionist Layered Object-oriented NEtwork Simulator (CLONES) is based on this interface and supports training and interconnect of a variety of network types including arbitrarily shaped backpropagation networks [Koh91].

The matrix vector library is in turn implemented on top of another shared interface: the Common Server Interface (or CSI). Object classes (such as matrix and vector) that insulate the user from the hardware configuration are built on top of the CSI class. The CSI class defines the interface that is inherited by classes such as `CSI_SPARC`, `CSI_RAP` and `CSI_SPERT`. These three classes contain the implementation of CSI for each of the three hardware configurations. Multiple `CSI_RAP` or `CSI_SPERT` objects can be created allowing the host to command more than one RAP or SPERT system.

In support of the SPERT library class developer, a SPERT assembler and Sparcstation C++ run-time libraries will be provided. The run-time libraries will provide test, bootstrap, memory management, function call, and data I/O routines for the SPERT board.

A software simulation of the SPERT system will be provided to allow application development where SPERT hardware is not available.

7 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 backpropagation training and in this section we present detailed performance results for backpropagation 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 3 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 352×10^6 connections per second (CPS). There are nine instructions in the inner loop for forward propagation. The first instruction brings in a vector of 8 inputs; the subsequent 8 instructions load in 8 vectors of 8 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. The

loop is unrolled and software pipelined to sustain peak performance. SPERT attains high performance even with smaller nets. Nets must be smaller than 32×32 to drop below half peak performance.

Figure 4 plots the performance of SPERT for backpropagation training of input unit layer to hidden unit layer weights in a multilayer feedforward network. This include the time for forward propagation, and the time to calculate error terms and update weights during backpropagation. No errors are backpropagated in this case. Peak performance is around 114×10^6 connection updates per second (CUPS).

Figure 5 plots the performance of SPERT for backpropagation training of hidden unit layer to output unit layer weights in a multilayer feedforward network. This includes forward propagation, output error calculation, weight updates, and error backpropagation. Peak performance is around 86×10^6 CUPS.

In this case, performance drops off noticeably with fewer output units. This is due to the way in which errors are backpropagated. The backpropagate routine has an outer loop over groups of 8 hidden units, and an inner loop over groups of 8 output units. The errors for each hidden unit are accumulated as 8 partial error terms, one per datapath, in a vector register. When the inner loop completes, it is necessary to sum the 8 partial errors in a vector register for each of the 8 hidden units in the current iteration of the outer loop. This is performed in two stages. First, each set of 8 partial errors are reduced to 4 by using 4×32b memory accesses to move 4 partial sums to the other half of the SIMD array. Next, the scalar unit accesses the vector registers and performs three additions per vector register to reduce the 4 partial sums to a single error for each hidden unit. This set of summations must be performed after completing every inner loop over output units, and so represents a significant overhead with smaller numbers of output units. There are several possible ways to reduce this overhead. A future SPERT design could implement a vector reduction unit that sums the 8 elements in a vector register using a carry save array. Although this would provide a useful boost to training performance on smaller networks, it has not been included in the current implementation to save design time and complexity.

Combining the above results we obtain Figure 6. This 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 100×10^6 CUPS.

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 updates occuring 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.

8 Related Work

Several related efforts are underway to construct programmable digital neurocomputers, most notably the CNAPS chip from Adaptive Solutions [Ham90] and the MA-16 chip from Siemens [RBR+91].

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, and 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.

In summary, we are designing SPERT to provide the large off-chip memory bandwidth of the Siemens chip as well as the general programmability of the Adaptive Solutions chip. SPERT also 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, subnetworks. 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 impor-

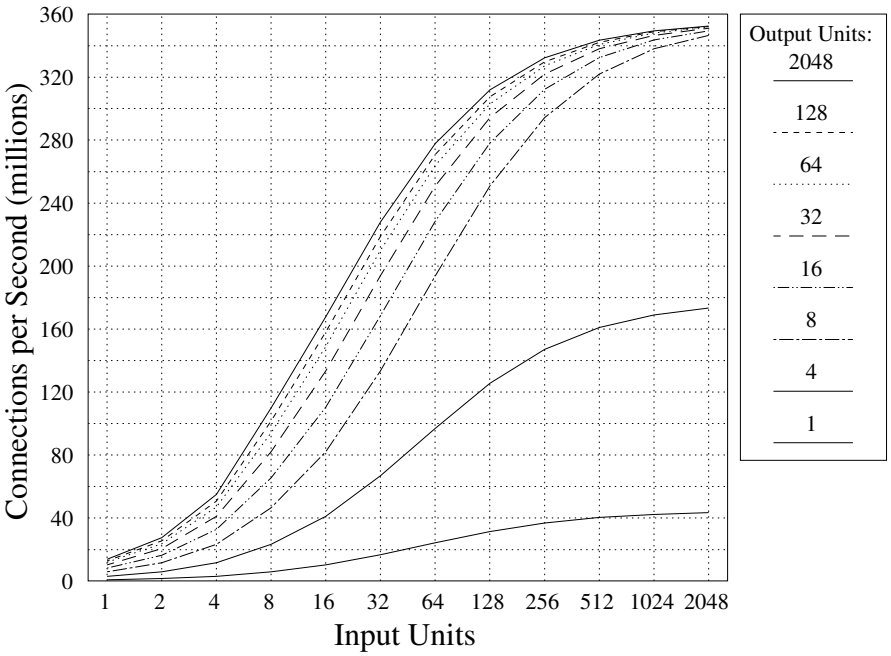

Figure 3: SPERT Forward Propagation Performance.

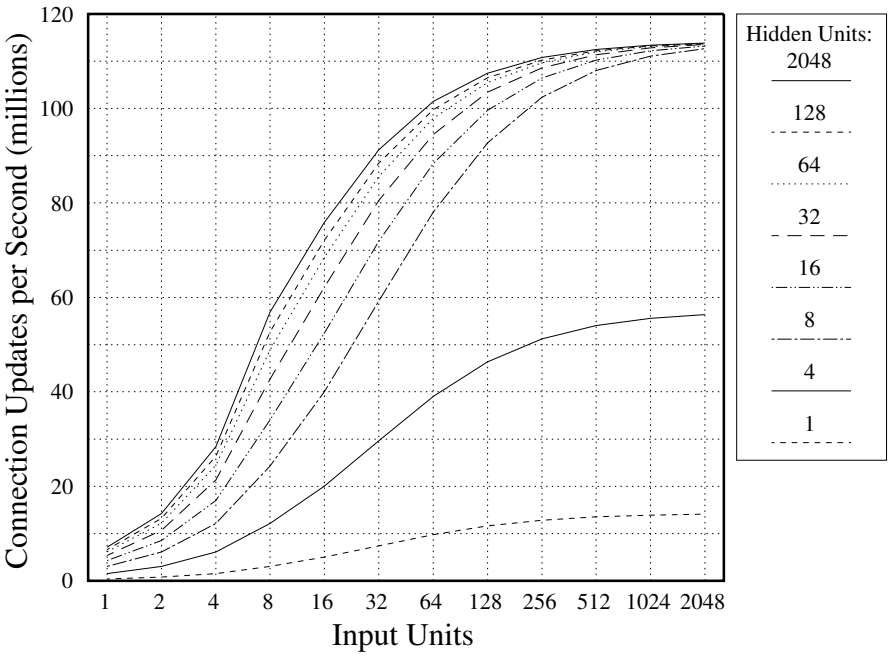

Figure 4: SPERT Training Performance. Input Unit to Hidden Unit Weights.

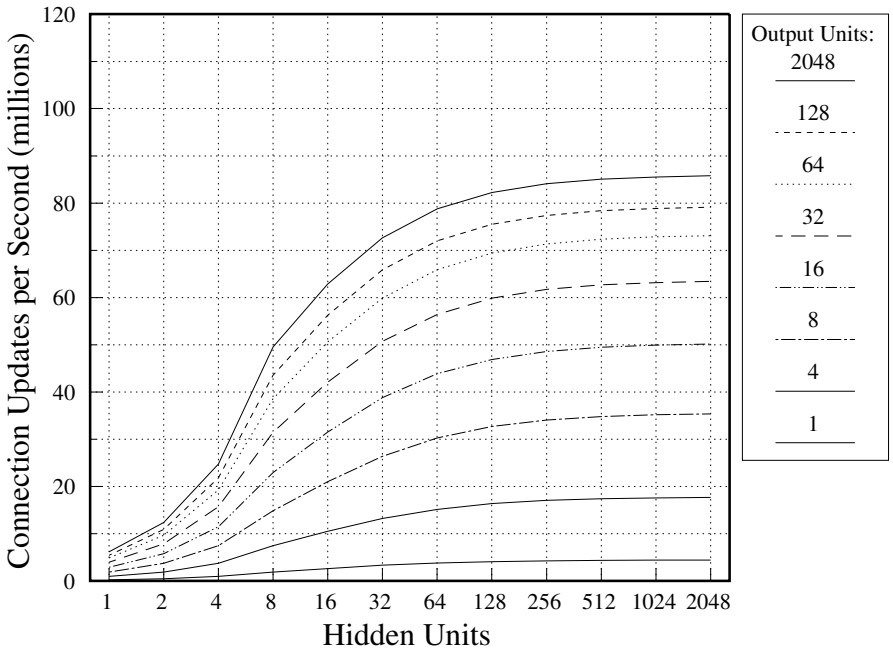

Figure 5: SPERT Training Performance. Hidden Unit to Output Unit Weights.

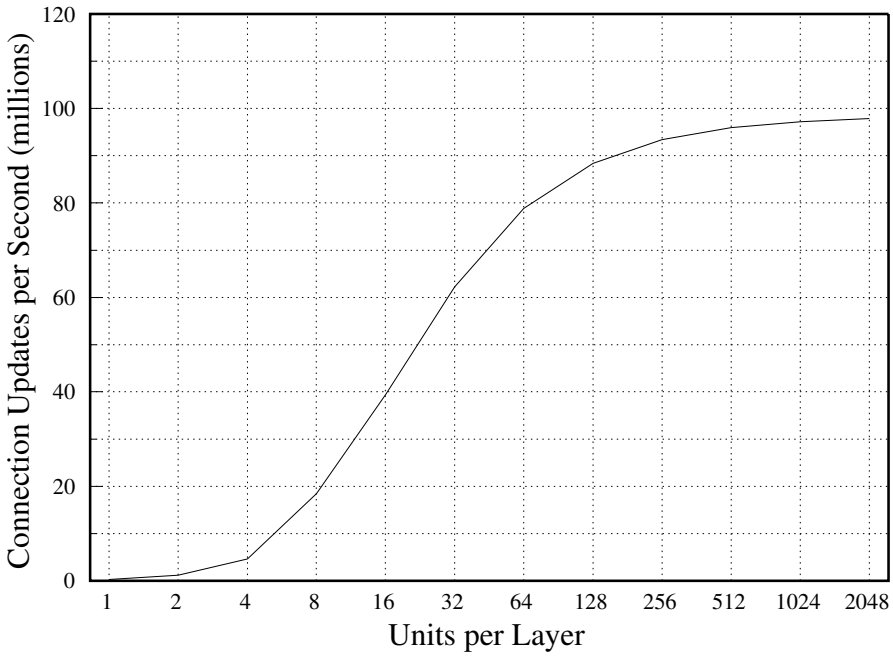

Figure 6: SPERT Training Performance. Three layers, with equal number of units per layer.

tant 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 will be targeted at high performance on large, irregular network structures.

9 Summary

We have presented an overview of the SPERT project. For those connectionist applications that do not require high precision arithmetic, SPERT represents a very high performance, low cost acceleration tool. The predicted sustained performance is over 300×10^6 CPS for forward propagation, and around 100×10^6 CUPS for error backpropagation in layered feedforward networks. A consistent set of software abstractions unify SPERT with our other systems, including the earlier RAP, and a larger multiprocessor we are currently designing with further custom VLSI elements based on SPERT.

10 Acknowledgements

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

[AM91] Krste Asanović and Nelson Morgan. Experimental Determination of Precision Requirements for Back-Propagation Training of Artificial Neural Networks. In *Proceedings 2nd International Conference on Microelectronics for Neural Networks*, Munich, October 1991.

[ASPF92] Krste Asanović, Klaus Erik Schauser, David A. Patterson, and Edward H. Frank. Evaluation of a Stall Cache: An Efficient Restricted On-Chip Instruction Cache. In *Proceedings 25th Hawaii International Conference on System Sciences*, January 1992.

[BH88] Tom Baker and Dan 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.

[Ham90] Dan Hammerstrom. A VLSI architecture for High-Performance, Low-Cost, On-Chip Learning. In *Proc. International Joint Conference on Neural Networks*, pages II–537–543, 1990.

[KAW+91] Brian E. D. Kingsbury, Krste Asanović, John Wawrzynek, Bertrand Irissou, and Nelson Morgan. Recent Work in VLSI Elements for Digital Implementations of Artificial Neural Networks. Technical Report TR-91-074, International Computer Science Institute, 1991.

[Koh91] P. Kohn. CLONES: Connectionist Layered Object-oriented NEtwork Simulator. Technical Report TR-91-073, International Computer Science Institute, 1991.

[MB90] N. Morgan and H. Bourlard. Continuous speech recognition using Multilayer Perceptrons with Hidden Markov models. In *Proc. IEEE Intl. Conf. on Acoustics, Speech, & Signal Processing*, pages 413–416, Albuquerque, New Mexico, USA, 1990.

[MBAB90] N. Morgan, J. Beck, E. Allman, and J. Beer. RAP: A Ring Array Processor for Multilayer Perceptron Applications. In *Proc. IEEE Intl. Conf. on Acoustics, Speech, & Signal Processing*, pages 1005–1008, Albuquerque, New Mexico, USA, 1990.

[MBKB92] N. Morgan, J. Beck, P. Kohn, and J. Bilmes. Neurocomputing on the RAP. In K. W. Przytula and V. K. Prasanna, editors, *Digital Parallel Implementations of Neural Networks*. Prentice Hall, 1992. In Press.

[RBR+91] 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*. Kluwer Academic, 1991.

[RHW86] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning Internal Representations by Error Propagation. In *Parallel Distributed Processing. Exploration of the Microstructure of Cognition*, volume 1. MIT Press, 1986.

[Tex88] Texas Instruments, Houston, Texas, USA. *Third-Generation TMS320 User's Guide*, 1988.

[Waw92] J. Wawrzynek. A 250MHz 64-bit Datapath in 1.2 μm CMOS. Technical Report In Preparation, Computer Science Division (EECS), University of California, Berkeley, 1992.

[Wer74] P.J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* PhD thesis, Dept. of Applied Mathematics, Harvard University, 1974.

[YS89] Jiren Yuan and Christer Svensson. High-Speed CMOS Circuit Technique. *IEEE JSSC*, 24(1):62–70, February 1989.