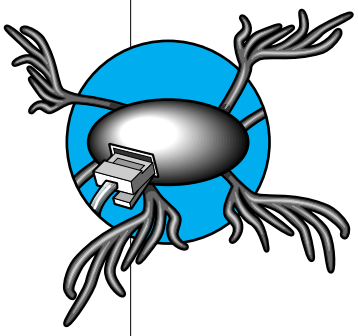


Spert-II: A Vector Microprocessor System

John Wawrzynek
Krste Asanović
Brian Kingsbury
University of California, Berkeley

David Johnson
James Beck
Nelson Morgan
International Computer Science Institute

The Spert-II fixed-point vector microprocessor system performs training and recall faster than commercial workstations for neural networks used in speech recognition research.

Many algorithms used in human-machine interface applications—such as speech, image, and handwriting recognition—require extensive computation. The burgeoning area of multimedia processing also requires high-performance computing.

Processing elements for use in these areas must provide high performance at low cost, as they probably will appear in a variety of consumer products. The processing elements should also be programmable, to support multiple standards and reduce new algorithms' time-to-market. A programmable device can also be used in multiple applications, which makes it easier to recover silicon development costs.

Any programmable device brings associated software development costs. Ideally, future processing-element implementations will be object-code-compatible with earlier designs, so users can preserve their software investments while taking advantage of fabrication technology advances that make the code run faster.

We believe vector microprocessor architectures will make ideal processing elements for these multimedia and human-machine interface applications, which often contain algorithms that can be expressed in data-parallel form. Also, moderate-precision fixed-point arithmetic is often sufficient, which lets us integrate many parallel functional units on one die. A vector instruction-set architecture (ISA) allows a natural expression of the applications' data parallelism and simplifies implementations that employ multiple parallel units and pipelined functional units.

We have packaged a prototype full-custom vector microprocessor, T0, as the Spert-II (Synthetic Perceptron Testbed II) workstation accelerator system (see the sidebar "Vector microprocessors"). We originally developed Spert-II to accelerate multiparameter neural network training for speech recognition research.[1] Our speech research algorithms constantly change. Also, neural nets are often integrated with other tasks to form complete applications. We thus desired a general-purpose, easily programmable accelerator that could speed up a range of tasks.

THE TORRENT VECTOR MICROPROCESSOR

The T0 vector microprocessor's development follows our earlier work on the Spert VLIW/SIMD (very long instruction word/single-instruction multiple-data) neuromicroprocessor.[2] We have since moved to a vector ISA that is based on the industry-standard MIPS RISC scalar ISA[3] extended with vector coprocessor instructions. The resulting ISA, which we call Torrent, offers important advantages over our previous design. We gain

0018-9162/96/$5.00 © 1996 IEEE

access to scalar MIPS architecture software tools, including optimizing C and C++ compilers, assemblers, linkers, and debuggers. The vector ISA reduces instruction fetch bandwidth and will enable object-code compatibility with future designs.

The T0 processor is a complete single-chip Torrent architecture implementation fabricated by Hewlett-Packard's CMOS26G process using 1.0-μm scalable CMOS design rules and two metal layers. The die measures 16.75 mm × 16.75 mm and contains 730,701 transistors. At a clock frequency of 40 MHz, the device consumes about 12W of power from a 5V supply. The first silicon, received in April 1995, is fully functional with no known bugs.

T0 achieves high performance at low cost by integrating multiple fixed-point data paths with a high-bandwidth memory system. Fast digital arithmetic units, such as multipliers and shifters, require chip area proportional to the square of the number of operand bits. In modern microprocessors and digital signal processors, a floating-point unit occupies much of the chip. Higher-precision arithmetic units also need higher memory bandwidth to move large operands. However, many problems do not require full-precision floating-point or even high-precision fixed-point arithmetic. Studies have shown that for error back-propagation neural network training, 16-bit weights and 8-bit activation values perform as well as 32-bit single-precision floating-point.[4]

However, fast fixed-point multiply-adds are not sufficient to increase performance on many problems. Other application components may dominate total computation time if only the multiply-add operations are accelerated. The vector unit includes general-purpose operations in its instruction set and is tightly coupled to a fast general-purpose RISC core to handle nonvectorizable operations.

As Figure 1 shows, T0's main components are the MIPS-II-compatible RISC CPU with a 1-Kbyte on-chip instruction cache, a vector unit coprocessor, an external memory interface, and an 8-bit-wide serial host interface (TSIP) and control unit. The external memory interface supports up to 4 Gbytes of memory over a 128-bit-wide data bus. The system coprocessor provides a 32-bit counter/timer and registers for host synchronization and exception handling.

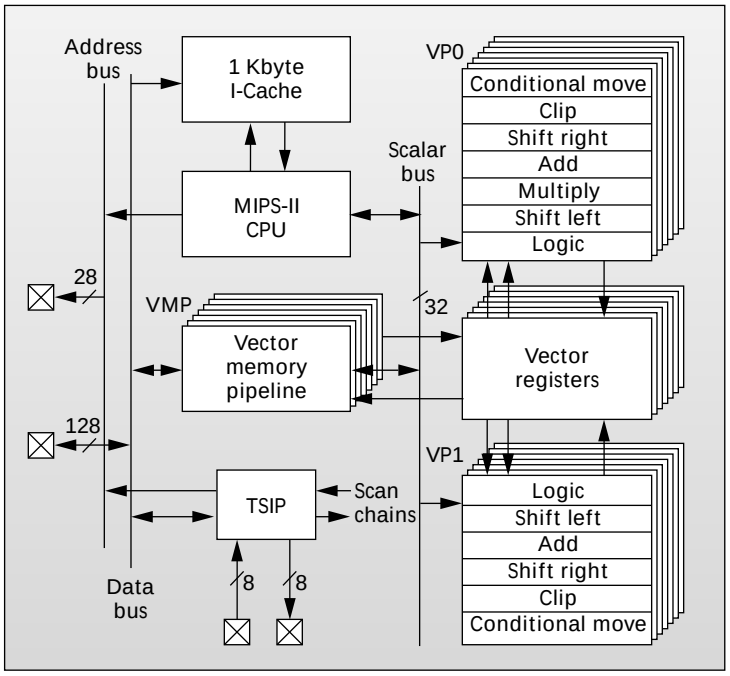


Figure 1. Block diagram of T0 microarchitecture.

The vector unit contains a vector register file and the VP0, VP1, and VMP vector functional units. The vector register file contains 16 vector registers, each holding 32 32-bit elements. VP0 and VP1 are vector arithmetic functional units that can perform 32-bit integer arithmetic and logic operations and support fixed-point scaling, rounding, and saturation. Multiplication is supported only in VP0, with 16-bit × 16-bit multiplies producing 32-bit results. The software performs vector division by long division, using an estimate of the divisor's reciprocal to obtain 12 quotient bits per iteration. VMP, the vector memory unit, handles all vector load/store operations, scalar load/store operations, and vector insert/extract operations.

Vectors are addressed in external memory with three types of load/store options—

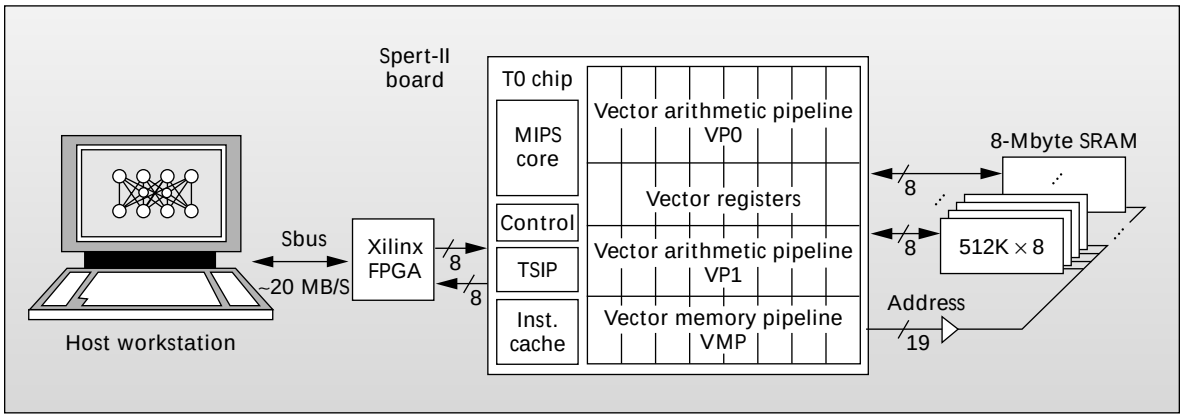


Figure 2. Spert-II system organization.

unit stride, nonunit stride, and indexed access. In unit-stride, vector elements occupy consecutive memory locations. In nonunit stride, elements are separated by a constant distance. With indexed access, a vector register provides pointers to the operand vector's elements. This option efficiently implements parallel table-lookup functions for function approximation. It also supports sparse matrix-vector operations.

Each vector functional unit is composed of eight parallel pipelines and produces up to eight results per cycle. The T0 memory interface has one memory address port, which limits nonunit stride and indexed memory operations to one element transfer per cycle.

A vector register's elements are striped across all eight pipelines. With a maximum vector length of 32, a vector functional unit accepts a new instruction every four cycles. T0 can saturate all three vector functional units by issuing one instruction per cycle to each, leaving the scalar unit an issue slot every four cycles. T0 can thus sustain up to 24 operations per cycle, a level achieved by several important library routines, such as matrix-vector and matrix-matrix multiplies. All vector pipeline hazards are interlocked in hardware, so instruction scheduling is not required for correctness but may improve performance.

THE SPERT-II WORKSTATION ACCELERATOR

Spert-II is a double-slot Sbus card for Sun-compatible workstations. As Figure 2 shows, the board contains a T0 vector microprocessor, SRAM, a Xilinx FPGA device for interfacing with the host, and various system support devices.

We use chip-on-board (COB) technology to mount the T0 die because COB provides lower cost and better electrical performance than conventional chip packaging. We fabricate the board using a conventional printed circuit board (PCB) process and glue the die into a cavity with thermally conductive epoxy. Thermal vias under the bond site conduct heat to the board's rear, where a heat sink and a fan are attached. Signal and power connections are made through conventional wire bonds. The PCB bond pads are interleaved on two board levels to match the board trace pitch of 300 μm to the die pad pitch of 150 μm. Figure 3 shows a T0 processor die that has been mounted on a Spert-II board.

Figure 4 shows a complete Spert-II board. On the top is a T0 processor, under a metal cover, surrounded by surface-mounted address drivers and SRAM parts. On the bottom is the board's opposite side, which has more memory parts, the Xilinx FPGA, clock control support devices, and space for the T0 heat sink. Spert-II boards have a 40-MHz processor clock frequency and use 16 20-ns, 4-Mbit SRAM parts to provide 8 Mbytes of main memory.

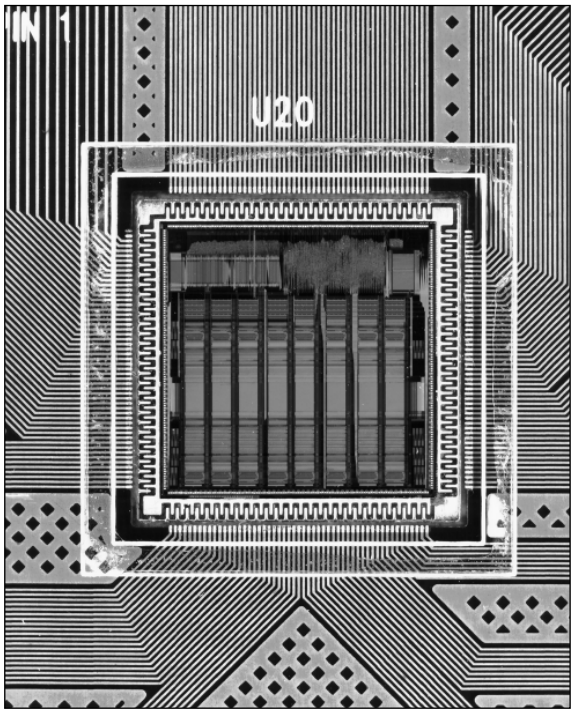
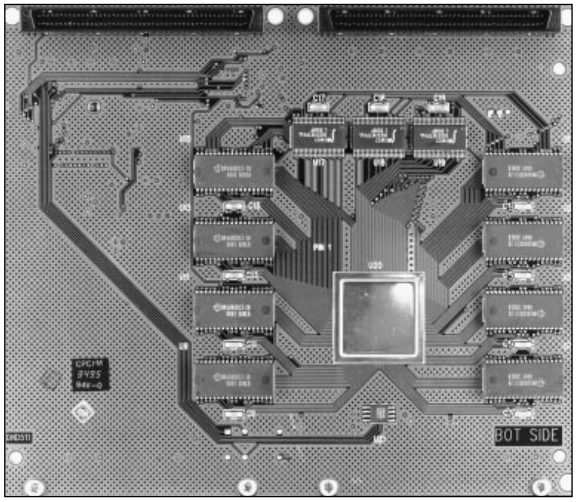


Figure 3. A T0 processor die that is mounted on a Spert-II board.



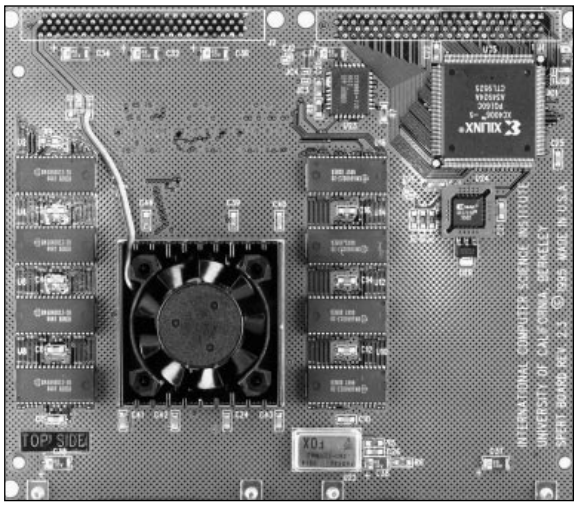


Figure 4. A complete Spert-II board. Top: On one side, a T0 processor, under a metal cover, is surrounded by address drivers and SRAM parts. Bottom: On the other side are memory parts, the Xilinx FPGA, clock control support devices,the T0 heat sink, and the fan.

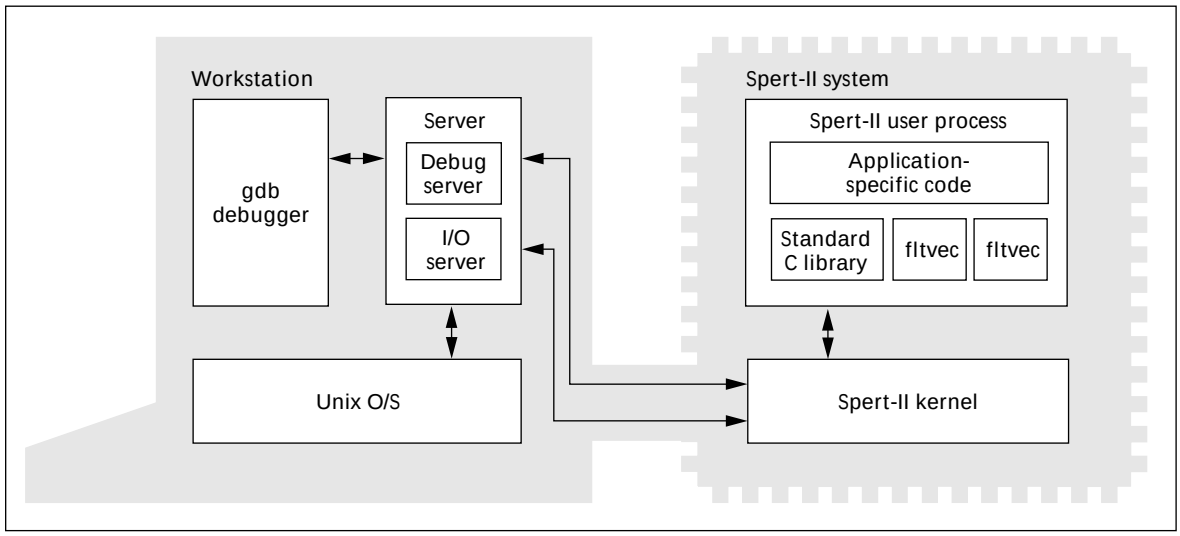


Figure 5. The Spert-II software environment.

SPERT-II SOFTWARE ENVIRONMENT

We wanted the Spert-II software environment, shown in Figure 5, to look like a conventional workstation software environment to ease the porting of existing workstation applications and to provide a comfortable code development environment.

We add the Torrent vector instructions as coprocessor instructions to the base MIPS-II instruction set. These operations use an opcode space section reserved in the MIPS architecture for such added functionality. This compatibility lets us base our software environment on the popular GNU tool set, which already supports MIPS-based machines.

We use an unmodified version of the *gcc* C/C++ compiler and modify the *gdb* symbolic debugger to debug T0 programs remotely from the host. We extend the *gas* assembler to support the new vector instructions and add an optimizing code scheduler that reorders instructions to reduce the number of hardware interlocks. We also employ the GNU linker and other binary utility programs, such as library archivers.

Vector unit access is either through library routines or by way of the scheduling assembler. We developed an extensive optimized vector library routine set, including fixed-point matrix and vector operations, function approximation through linear interpolation, and IEEE single-precision floating-point emulation. We wrote a parallel set of functions in ANSI C to permit Spert-II program development on workstations. Finally, a standard C library contains the usual utility, I/O, and scalar math routines.

The Spert-II board's operating system has two components—a small kernel that runs on T0 and a user-level server that runs on the host. To run a program on Spert-II, a user invokes the server, passing the Spert executable's name as an argument. The server resets T0, downloads the kernel and Spert executable, and waits to handle Spert program I/O requests.

T0 does not include a hardware floating-point unit, but it fully supports floating-point code. The MIPS-II floating-point instructions are trapped and emulated by the kernel, simplifying software porting. We can achieve greater throughput for vectorizable floating-point code by using the single-precision vector floating-point library.

MAPPING BACK PROPAGATION

We have implemented an artificial neural network (ANN) training task taken from a speaker-independent continuous speech recognition system. The ANN is a simple three-layer, feed-forward perceptron. Multilayer perceptrons for this task typically have 100 to 400 input units. The input layer is fully connected to a hidden layer of 100 to 4,000 units. The hidden layer is fully connected to an output layer containing between 56 and 61 units representing phonemes. The hidden units incorporate a standard sigmoid nonlinearity, typically $f(x) = 1/(1 + e^{-x})$. The output units compute a soft-max

$$f(x) = \frac{e^{x}}{\Sigma_i\, e^{x_i}}$$

or a sigmoid activation function.

Some researchers have experienced slow convergence when using back propagation to train small networks with limited data sets. However, we experience relatively fast convergence using on-line back propagation to train large networks with large real-world data sets for our speech recognition tasks. A randomly initialized net containing from 40,000 to over 1 million weights typically converges with three to 10 passes over the training database, which may contain several million pattern presentations. In some cases, a previously initialized network that is being adapted converges with only one pass.

We use on-line training with weight matrices updated after each pattern presentation. Other neurocomputers' benchmarks often use a batch update procedure, where training patterns are grouped together and weight matrices are updated only after processing an entire group. This is typically easier to parallelize than an on-line or per-pattern weight update, which requires communication for every pattern presentation. We have found that one pass of on-line training accomplishes more than one pass of

Vector microprocessors

Commercial microprocessor performance has increased dramatically, driven largely by CMOS fabrication technology improvements. Superscalar microprocessors currently execute up to 4 scalar instructions per cycle, with clock cycle times dropping as low as 3.3 ns.[1] Issuing more instructions in parallel brings diminishing returns, because the logic required to dynamically detect instruction stream parallelism and then communicate data values between concurrent instructions grows quadratically with the number of instructions issued per cycle. Current superscalar microprocessors already dedicate considerable die area to superscalar instruction issue management.

Very long instruction word (VLIW) processors have been proposed to reduce the control complexity of executing multiple operations per cycle.[2] On every cycle, the hardware implementation fetches and executes a VLIW instruction containing multiple operations. The VLIW compiler finds independent operations and packs them into VLIW instruction to satisfy intra-instruction and interinstruction dependencies. The VLIW hardware configuration is made visible at the instruction set level, so code compiled for one member of a VLIW machine family must be recompiled to run on another. This is a disadvantage, as are the large instruction cache space and instruction fetch bandwidth required by the long instruction words.

Vector processors, unlike conventional scalar processors, can specify multiple independent operations on linear operand arrays in one instruction.[3] For example, one vector addition instruction can read two vectors of equal length and add corresponding elements to produce a third results vector.

Many applications contain abundant data parallelism that can be readily mapped to vector instructions. A low-cost implementation may execute a vector addition's independent operations with one pipelined adder producing one result per cycle, as Figure A1 shows. A high-performance implementation may employ an array of *N* parallel pipelined adders producing *N* results per cycle, as Figure A2 shows. There is little control logic increase between these two implementations, and vector machines can be easily scaled to higher degrees of parallelism as fabrication technology allows.

Vector architectures have traditionally been employed in large supercomputers used for scientific and engineering tasks. We are investigating the use of inexpensive vector microprocessors on new computationally intensive tasks within commodity applications, such as multimedia processing and human-machine interfacing.

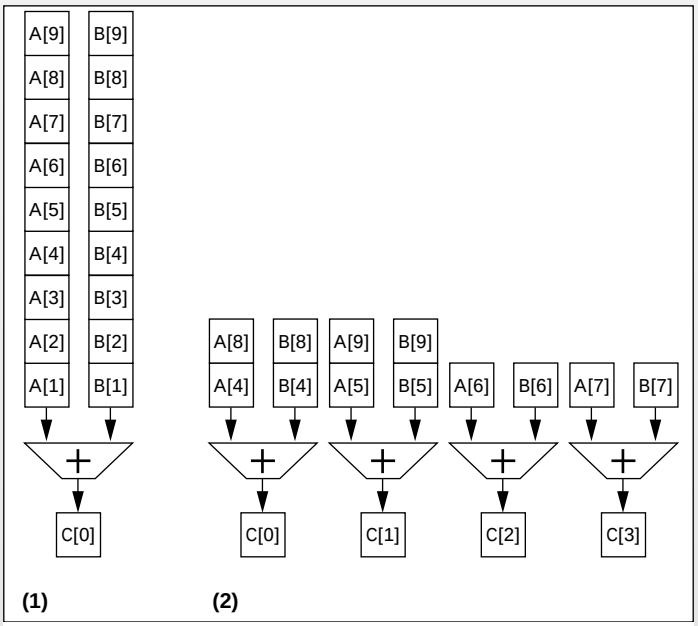


Figure A. A vector instruction-set-architecture can have a low-cost implementation that executes one operation per cycle or a high-performance implementation that executes multiple operations per cycle. The figures show a low-cost implementation (1) and a high-performance implementation (2) executing the addition of 10 elements of two vectors, A and B, to give a result vector, C. The high-performance version completes four results per cycle.

References

1. B.J. Benschneider et al., "A 300-MHz 64-b, Quad-Issue CMOS RISC Microprocessor," *IEEE J. Solid-State Circuits*, Vol. 30, No. 11, Nov. 1995, pp. 1,203-1,214.
2. J. Fisher, "Very Long Instruction Word Architectures and the ELI-512," *Proc. 10th Int'l. Symp. Computer Architecture*, IEEE CS Press, Los Alamitos, Calif., Order No. PR473, 1983, pp. 140-150.
3. J. Hennessy and D. Patterson, *Computer Architecture—A Quantitative Approach*, Morgan Kaufmann, Palo Alto, Calif., 1990, pp. 350-396.

batched training for large realistic problems, such as speech or image recognition, where data sets tend to be redundant.

We have named the programs that perform forward pass and training *qnforward* and *qntrain*, respectively. These programs are written in C++, using library routines to perform matrix and vector operations. The Spert-II system runs the network training program on the T0 processor. This contrasts with other neural network accelerators, in which users must partition user code to place general computations on the host and to place only the computationally intensive operations on the custom hardware. This partitioning adds coding effort, and the resulting communication increases costs.

We use the same source code for workstation and Spert-II training, except that the Spert-II MLP class calls fixed-point matrix-vector libraries, while the workstation MLP class calls floating-point matrix-vector libraries. The fixed-point and floating-point MLP classes have the same interface and use single-precision floating-point values to pass input and output values.

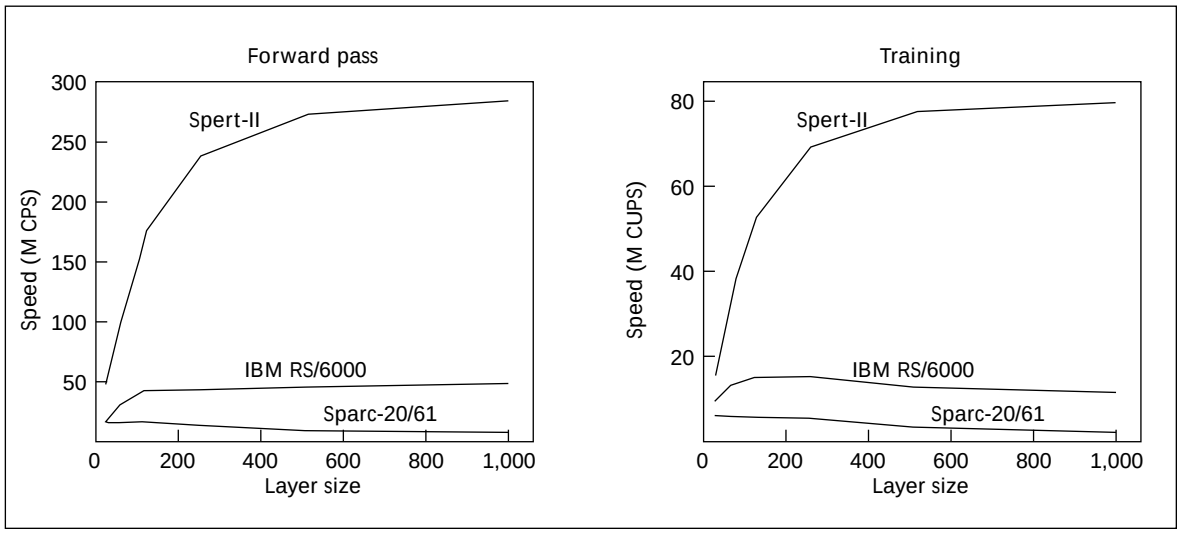


Figure 6. Performance comparison of the Spert-II and two commercial RISC workstations for a set of three-layer networks on forward-propagation and back-propagation training.

In addition to forward- and back-propagation functions, *qnforward* and *qntrain* perform such operations as training and cross-validation database management, database format conversion, control of learning schedules, checkpointing, and status reporting. We employ the same source code we use for these ancillary functions on the workstations and Spert-II. The *qntrain* and *qnforward* source code for workstations and Spert-II has about 8,000 lines, of which about 1,000 support the fixed-point version. These totals do not include the source code for the floating-point and fixed-point matrix-vector libraries.

The fixed-point matrix-vector library contains hundreds of functions, although we use only a limited subset for back-propagation training. The core computationally intensive operations in back-propagation training are forward propagation, error back propagation, and weight update. These map to vector-matrix multiply, matrix-vector multiply, and scaled outer-product accumulation, respectively. We highly optimize these routines and rearrange the loops to use unit-stride memory accesses. We reduce memory bandwidth requirements by tiling matrix accesses and reusing vector register operands when possible.

The library also contains many simpler vector operations. We use some to handle input and output vectors and activation values in back-propagation training. While these operations require only $O(n)$ computation, as opposed to matrix operations' $O(n^2)$ requirements, they increase costs significantly on small networks if not vectorized.

For example, we implement the sigmoid activation function by using a piecewise-linear function-approximation routine that uses vector-indexed load operations to perform table lookups. T0 can execute vector-indexed operations at only one element transfer per cycle. However, the table lookup routine can simultaneously perform all arithmetic operations for index calculation and linear interpolation in the vector arithmetic units at a rate of one 16-bit sigmoid result every 1.5 cycles. Similarly, we use a table-based vector logadd routine to implement the soft-max function at a rate of one result every 2.25 cycles.

T0 also uses vector library routines to convert single-precision IEEE format to the internal 16-bit fixed-point representation at the rate of 2.4 cycles per element converted.

PERFORMANCE EVALUATION

We compared the Spert-II system with two commercial RISC workstations. One was a Sparcstation-20/61 containing one 60-MHz SuperSparc+ processor with a peak performance of 60 million floating-point operations per second (Mflops), 1 Mbyte of second-level cache, and 128 Mbytes of DRAM main memory. The other was an IBM RS/6000-590 computer containing the RIOS-2 chip set running at 71.5 MHz with a 266-Mflop peak performance, 256 Kbytes of primary cache, and 768 Mbytes of DRAM main memory. The Spert-II system contains the T0 processor running at 40 MHz with a peak performance of 640 million fixed-point operations per second and 8 Mbytes of SRAM main memory, mounted in a Sparcstation-5/70.

Figure 6 shows the three systems' performance for a set of three-layer networks on forward-propagation and back-propagation training. Table 1 presents performance results for two speech-network architectures. We used three-layer neural networks with total connectivity between adjacent layers. In Figure 6, for ease of presentation, we used networks with the same number of units per layer, beginning at 32 and increasing by powers of two. The networks presented in Table 1, however, had a different number of units per layer. The sigmoid function was the hidden-layer activation function, and the soft-max function was the output-layer activation function.

The workstation version performs all input, output, and computation using IEEE single-precision floating-point arithmetic. We extensively hand-optimize the matrix and vector operations within the back-propagation algorithm using manual loop unrolling with register and cache blocking. The Spert-II timings include the time spent converting between floating-point and fixed-point for input and output.

The workstation code uses calls to library exponential routines to evaluate the activation functions, while the

Table 1. Performance of three microprocessor systems for two speech-network architectures.

Net type	Net size (in × hidden × out)	Spert-II	Sparc20	IBM RS/6000-590
Forward pass (M CPS)				
Small speech net	153 × 200 × 56	181.0	17.60	43.0
Large speech net	342 × 4,000 × 61	276.0	11.30	45.1
Training (M CUPS)				
Small speech net	153 × 200 × 56	57.1	7.00	16.7
Large speech net	342 × 4,000 × 61	78.7	4.18	17.2

Spert-II version uses linear interpolation from lookup-table values. The Sparcstation-20/61 system's workstation code spent slightly less than 6 percent of its time in the small speech network's activation function code and just over 1 percent in the large speech network's code. For large networks, the Spert-II system performed 30 times better than the Sparcstation-20/61 and about six times better than the IBM RS/6000-590. The Spert-II system clearly performed better even for small networks, such as those with only 32 units per layer.

We used a phoneme classification problem to compare how well the fixed-point and floating-point nets learn. Each net is trained to classify a string of acoustic feature vectors into phoneme categories. From a subset of the Bellcore spoken digits database, we use 1,720 sentences for back-propagation training and 230 for cross validation. We repeated training runs a number of times and used different random seeds to initialize the weights. The results, in Table 2, show that there is little difference in error rates and that the fixed-point net trains in an average of one fewer pass over the training database.

RELATED WORK

Previous programmable digital neurocomputers include systems based on the Adaptive Solutions CNAPS-1064 chip[5]; the Synapse-1, based on the Siemens MA-16 chip;[6] and our RAP (Ring Array Processor) system.[7]

The CNAPS-1064 is a SIMD device with 64 16-bit fixed-point processing nodes (PNs) on a single die. It requires an external control sequencer chip, possibly controlling multiple chained parts, to form a complete system. The chips run at 25 MHz and have a peak performance of 1,600M CPS and 250M CUPS. Each PN contains 4 Kbytes of local memory. The CNAPS system obtains high performance only with algorithms that can map data structures into each PN's local memory. The bandwidth to external memory is limited to 8 bits of input and 8 bits of output per cycle for each system chip.

The Synapse-1 system is formed by cascading the MA-16 into a large systolic array and adding considerable support logic. Each MA-16 contains 16 multiply-add units and peripheral logic, and each connects to off-chip DRAM for weight storage. Synapse-1 is a special-purpose system designed primarily to execute large matrix-multiplies and thus performs best with batch-mode training. The system has multiple control flow levels, and data must be carefully distributed across several disjoint memory spaces.

The RAP machine was built from off-the-shelf TMS320C30 digital signal processors (DSPs) arranged in a ring through an FPGA-based interconnect. We measured a 40-node system at up to 100M CUPS for an on-line back-propagation learning task. Current floating-point DSPs perform about three times better than DSPs used in the RAP, but the Spert-II board would still be considerably less expensive than an updated RAP design with equivalent performance.

The new TMS320C80 MVP processor's[8] DSP has roughly the same die size and clock rate as the T0 but is fabricated in a newer 0.55-μm process with three metal layers. The MVP contains four independent VLIW DSPs and a proprietary RISC processor. Peak throughput is four 16-bit, fixed-point multiply-adds per cycle with a 64-bit data bus for MVP, as opposed to eight 16-bit fixed-point multiply-adds per cycle with a 128-bit data bus for T0.

We believe these other systems are more difficult to program than T0. The CNAPS system has small local memories on each processing element and limited off-chip bandwidth to a separate large DRAM on the control sequencer. The Synapse-1 has several disjoint memory spaces for matrix operands and several different types of control flow. The RAP has multiple independent control threads, usually used in single-program multiple-data fashion. The RAP's DSPs can communicate only by way of the ring, and each DSP has multiple on-chip memory banks, off-chip SRAM, and off-chip DRAM. For peak performance, we must carefully distribute operands among these memory spaces. The TMS320C80 has five control

Table 2. Fixed-point and floating-point network training quality and rate for phoneme classification. An epoch is one pass over the training database.

	Floating-point	Fixed-point
Training runs	28	82
Minimum percentage of frames correct	82.52	82.56
Mean percentage of frames correct	83.62	83.48
Maximum percentage of frames correct	84.70	84.37
Standard deviation	0.55	0.44
Minimum epochs training	6	5
Mean epochs training	7.68	6.85
Maximum epochs training	10	9

threads. We must explicitly move data between on-chip and off-chip memories using a separate DMA engine to attain peak memory bandwidth. In contrast, T0 has a single control thread and a single memory space.

WE HAVE PRESENTED A WORKSTATION ACCELERATOR based on a unique custom vector microprocessor and have described how we applied the system to neural network training for a speech recognition task. We have shown that many of neural network training's computationally intensive matrix operations, as well as many other operations, can be readily vectorized. We have also demonstrated that fixed-point representation is adequate for learning a phoneme recognition task.

Vector processors are perhaps the simplest and most efficient architectures for exploiting fabrication technology advances. Meanwhile, it appears likely that many new applications, such as those in multimedia and other human- machine interfaces, will be readily vectorizable. We thus believe vector microprocessors will prove to be increasingly important. ■

Acknowledgments

Thanks to Jerry Feldman for his contributions, Bertrand Irissou for his work on the T0 chip, John Hauser for Torrent libraries, John Lazzaro for his advice on chip and system building, and the anonymous reviewers for their comments on earlier drafts of this article. Primary support for our work came from ONR URI Grant N00014-92-J-1617, ARPA Contract N0001493-C0249, NSF Grant MIP-9311980, and NSF PYI Award MIP-8958568NSF. Additional support was provided by ICSI. IBM donated the RS/6000.

References

1. N. Morgan and H. Bourlard, "An Introduction to Connectionist Speech Recognition," *Signal Processing Magazine*, Vol. 12, No. 3, May 1995, pp. 25-42.
2. J. Wawrzynek, K. Asanović, and N. Morgan, "The Design of a Neuromicroprocessor," *IEEE Trans. Neural Networks*, Vol. 4, No. 3, 1993, pp. 394-399.
3. G. Kane and J. Heinrich, *MIPS RISC Architecture*, Prentice Hall, Englewood Cliffs, N.J., 1992.
4. K. Asanović and N. Morgan, "Experimental Determination of Precision Requirements for Back-Propagation Training of Artificial Neural Networks," *Proc. Second Int'l. Conf. Microelectronics for Neural Networks*, Kyrill and Method Verlag, Munich, Germany, 1991, pp. 9-16.
5. D. Hammerstrom, "A VLSI Architecture for High-Performance, Low-Cost, On-Chip Learning," *Proc. Int'l. Joint Conf. Neural Networks*, IEEE Neural Networks Council, Ann Arbor, Mich., 1990, pp. II-537-II-543.
6. U. Ramacher et al., "Design of a First Generation Neurocomputer," in *VLSI Design of Neural Networks*, U. Ramacher and U. Ruckert, eds., Kluwer Academic, London, 1991, pp. 271-310.
7. N. Morgan et al., "The Ring Array Processor (RAP): A Multiprocessing Peripheral for Connectionist Applications," *J. Parallel and Distributed Computing*, Vol. 14, No. 3, Mar. 1992, pp. 248-259.
8. "TMS320C80 Multimedia Video Processor," tech. brief, Texas Instruments, Dallas, 1994.

John Wawrzynek *is an associate professor of electrical engineering and computer sciences at the University of California, Berkeley, where he teaches courses in computer architecture and VLSI system design. His research involves custom VLSI for signal processing and parallel computation. He holds a PhD in computer science from the California Institute of Technology and an MS in electrical engineering from the University of Illinois, Urbana-Champaign.*

Krste Asanović *is working toward a PhD in computer sciences at the University of California, Berkeley. From 1983 through 1989, he worked at the GEC Hirst Research Center, London, UK. His research interests include computer architecture, vector processing, VLSI design, and high-performance computing. He received a BA in electrical and information sciences in 1987 from Cambridge University, UK. He is a student member of IEEE and ACM.*

Brian Kingsbury *is working toward a PhD in computer sciences at the University of California, Berkeley. His research interests include VLSI design, digital-signal processing, and robust speech recognition. In 1989, he received a BS in electrical engineering from Michigan State University and a National Science Foundation Graduate Fellowship. Kingsbury is a member of Tau Beta Pi.*

David Johnson *is a senior staff software engineer at the International Computer Science Institute (ICSI), Berkeley, California. He has worked on embedded-communication and -control projects in industrial and academic environments. His primary research field is high-performance systems for signal-processing applications. He has an MA in computer science from the University of Cambridge, UK.*

James Beck *has been an ICSI staff engineer since 1989 and has built fast hardware for artificial neural net computation. In 1990, he designed a DSP-based machine, the RAP, which has been used at ICSI and other research sites. Beck is responsible for system and board design, as well as manufacturing and test design, on the Spert-II project. He earned a BS in 1971 from the California Institute of Technology.*

Nelson Morgan *is an adjunct professor in electrical engineering and computer sciences at the University of California, Berkeley, and an ICSI research scientist and Realization Group leader. He joined ICSI in 1988 and the University of California, Berkeley, in 1991. His interests include the design of algorithms, architectures, and systems for parallel signal processing and pattern recognition systems, particularly using connectionist and perceptually-based paradigms. Morgan received a BS in 1977, an MS in 1979, and a PhD in 1980 in electrical engineering and computer sciences from the University of California, Berkeley. He is a senior IEEE member and has five patents relating to different aspects of signal processing and pattern recognition. He has written two books and about 120 technical papers.*

Readers can contact Wawrzynek, Asanović, and Kingsbury at the University of California, Berkeley, e-mail {johnw, krste, bedk}@cs.berkeley.edu. Readers can contact Johnson, Beck, and Morgan at ICSI, e-mail {davidj,beck, morgan}@icsi.berkeley.edu.