# The Design of a Neuro-Microprocessor

John Wawrzynek, Krste Asanović, and Nelson Morgan, *Senior Member, IEEE*

*Abstract*— **This paper presents the architecture of a neuro-microprocessor. This processor was designed using the results of careful analysis of our set of applications and extensive simulation of moderate-precision arithmetic for back-propagation networks. We present simulated performance results and test-chip results for the processor. This work is an important intermediate step in the development of a connectionist network supercomputer.**

## I. Introduction

We are engaged in the development of a connectionist network supercomputer. This computer is targeted for a peak performance of $10^{11}$ connections per second for networks with up to a million units and a billion connections. It will include specialized sensor and actuator interfaces for interaction with real-time processes. We feel that this combination of large networks and real-world interaction is critical to the development of large neural network applications and has the potential of enabling new science.

Using implementation technologies available within the next several years, a system of this capacity is by necessity a multiprocessor. The design of such a machine is similar to that of a general purpose parallel computer. Both share the same set of issues relating to system partitioning and communication. Today's digital technology offers a mature implementation strategy for such systems. Future systems may employ analog or emerging new technologies, however, at this stage we are taking advantage of many years of digital processor design methodology and fabrication processes. We are focusing our systems efforts on digital implementations.

As an intermediate step in the development of a connectionist network supercomputer, we have designed a single chip CMOS neuro-microprocessor that provides significant improvements in cost/performance over earlier systems for artificial neural network calculations. The architecture is fully programmable, and efficiently executes a wide range of connectionist computations. Special emphasis has been placed on the support of variants of the commonly used back-propagation training algorithm for multilayer feed-forward networks [6], [7]. This class of networks is of interest in the speech recognition task that is the focus of our applications work [3].

An earlier system, the Ring Array Processor (RAP), was a multiprocessor based on commercial DSP's with a low-latency ring interconnection scheme [4]. We have used the RAP to simulate variable precision arithmetic and guide us in the design of the neuro-microprocessor [1]. 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, for the speech recognition training problems we have examined, back-propagation training algorithms only require moderate precision. Specifically, 16 b weight values and 8 b output values were sufficient to achieve training and classification results comparable to 32 b 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 datapaths per processor. Also, reduced precision operands make more efficient use of valuable processor-memory bandwidth.

Our experience with the RAP has also shown us that close interaction with real user applications is critical for neurocomputer design. There is a strong push from users to reduce the wall-clock time for their applications. Successful neural system design must be sensitive to what could be considered a "neural" corollary to Amdahl's Law:

> A connectionist accelerator can be best speed up an application by a factor of 1/(fraction of nonconnectionist computation).

For the applications we have observed, this speedup is in the range of 10 to 100. For this reason, we believe that general computational support is required even in a specialized connectionist machine, e.g., at least one general purpose processor per 10–100 neural multiply-accumulate units. Furthermore, memory capacity and bandwidth must keep pace with the rest of the machine to support real applications. Finally, regardless of its potential for speedup, any machine will go unused unless usable and efficient software is available. The RAP was a successful compromise of these concerns; it is used on a daily basis by a growing user's group at a number of research sites. SPERT is our first attempt to incorporate more specialized VLSI into a system in a equally balanced way.

## II. SPERT Architecture

SPERT is a processor design that combines a general purpose integer datapath, similar to those found in RISC microprocessors, a vector unit comprising a SIMD array of

Manuscript received August 17, 1992; revised September 23, 1992. The National Science Foundation provided support for the VLSI building blocks and design tools under Grant MIP-8922354. B. Kingsbury was supported by a Graduate Fellowship. J. Wawrzynek was supported by the National Science Foundation through the Presidential Young Investigator (PYI) Award, MIP-8958568. The larger project was supported by the ONR URI N00014-92-J-1617 and the International Computer Science Institute.

J. Wawrzynek is with the Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA 94704.

K. Asanović and N. Morgan are with the University of California, Berkeley and the International Computer Science Institute, Berkeley, CA 94704.

IEEE Log Number 9207193.

datapaths optimized for neural network computations, and a wide connection to external memory. The RISC-like datapath is used for general scalar computation, to provide loop and branch control, and to generate memory addresses. It is the moderate precision requirement of neural network calculations that permits SIMD processing at the chip level.

SPERT has a single 128 b VLIW instruction format. The instruction contains a number of orthogonal fields that control different functional units. The fields in the VLIW instruction are split into five major groups: host synchronization control, scalar datapath control, SIMD datapath control, external memory control, and an immediate field. The immediate field is used either as a value for the scalar datapath or as an absolute branch address. The instruction pipeline has seven stages, and, in the absence of instruction cache misses and host memory accesses, one instruction can be completed every cycle.

The overall structure of SPERT is shown in Fig. 1. The main components are a scalar 32 b integer datapath, a SIMD array containing eight 32 b fixed point datapaths, an instruction fetch unit with an instruction cache, the 128 b wide external memory interface, and a JTAG[1] interface and control unit.

SPERT functions as an attached processor for a conventional workstation host. The 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. A minimal system configuration will include a SPERT processor, a JTAG connection to a host processor, and external SRAM. The initial implementation of SPERT has a maximum clock frequency of 50 MHz. The external memory interface supports up to 16 MB of single cycle memory over a 128 b data bus. At the maximum 50 MHz clock rate, memory bandwidth is 800 MB/s. The 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.

The computation core of SPERT comprises the scalar unit and the SIMD array. The scalar unit, shown in detail in Fig. 2 contains a triple-ported general purpose register file, a 32 b adder (add0), a 32 b logical unit (lu), a branch comparator (brc), a 32 b shifter, the SIMD shift amount register (sa), and two address generation units (add1 and add2). These are connected by three buses (wbus, xbus, and ybus) managed by register bypassing hardware. The 32 b wide global scalar bus, scbus, is used to transfer data between the scalar unit and the rest of SPERT. It connects all SIMD datapaths to the scalar unit. The bus transfer two values per cycle.

The scalar unit connects to the external memory data bus through the memory latches in the SIMD datapaths. After an external memory load operation, the 128 b of data are held in latches in the SIMD array. A scalar load then selects one of the datapaths to broadcast the value it received over the scbus.

[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.

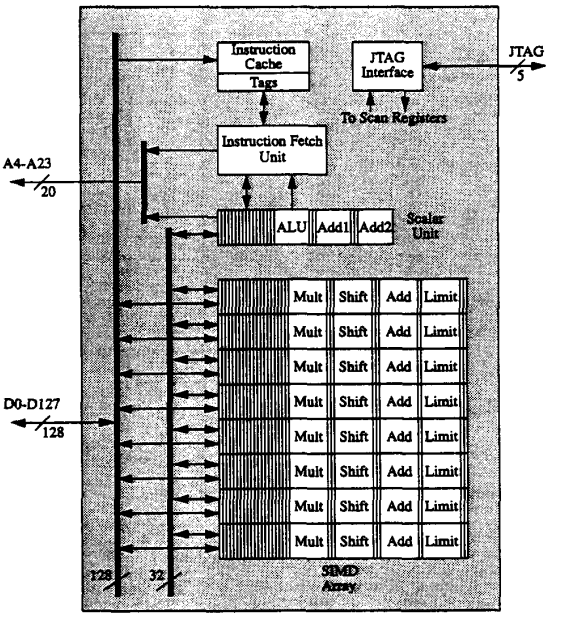


Fig. 1. SPERT structure.

This value can then be read by the scalar unit and/or all the SIMD datapaths. The same circuitry is used when performing a scalar unit load from SIMD register; the complete 256 b vector register is read by all SIMD datapaths, and only the selected datapath broadcasts the values to the rest of the system.

Fig. 3 shows the structure of one of the eight parallel SIMD datapaths. Each SIMD datapath is similar to that of a fixed-point DSP and contains 24 b × 8 b multiplier (mul), a 32 b saturating adder (sadd), a 32 b shifter (shift), and a 32 b limiter (lim). All datapaths in the SIMD array execute the same operation under control of a number of fields in the current instruction. All of these functional units can be active simultaneously, delivering up to $400 \times 10^6$ fixed-point multiply-accumulates per second. The multiplier produces a full 32 b result, but can optionally round the result to 24 b, 16 b, or 8 b. Both inputs to the multiplier can be treated as either signed or unsigned to support higher precision multiplies in software.

The functional units are connected to each other, the register file, and memory by three buses (a, b, and c). The buses transfer two operands per clock, to allow up to six values to be transferred per instruction. In addition, there are four sneak paths between physically adjacent functional units. In this manner, up to 10 values can be moved between functional units in every datapath each cycle. Over all eight datapaths, this is an operand bandwidth of up to 16 GB/s.

Each of the functional units has a register associated with each input. The input registers can be loaded with values from the register file, the output of other functional units, or can be left unchanged. They allow multiple operands to be transferred between functional units less expensively than providing multiple ports into a single shared register file. The registers can retain values across multiple instructions,

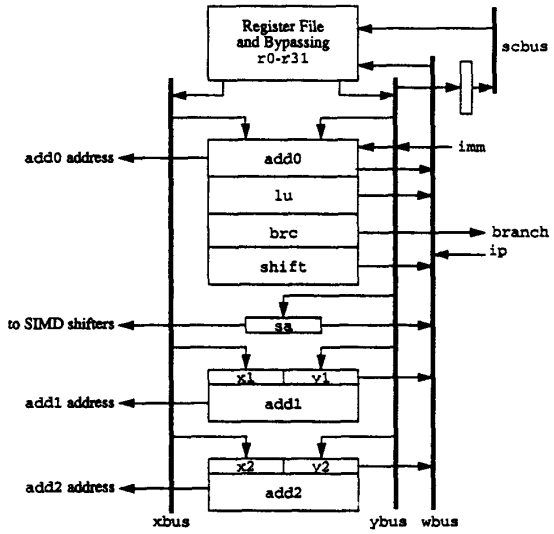


Fig. 2. Scalar unit datapath.

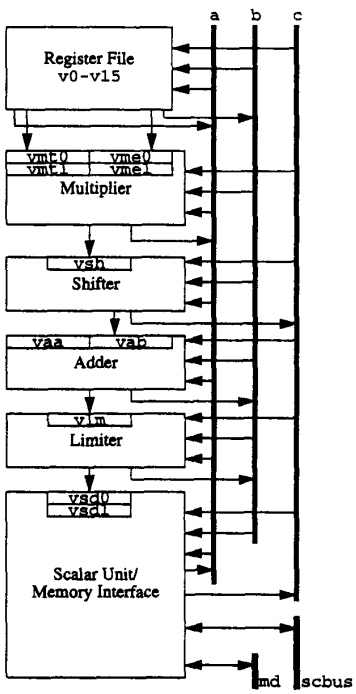


Fig. 3. SIMD datapath.

allowing constants to be held in the datapath with no further datapath bus traffic.

SPERT is a pure load/store architecture; all memory transfers are to/from registers, and all functional unit operations take register operands. All address generation is performed by the scalar unit. Memory transfers for the SIMD array can either move a vector of four or eight operands, or load a single scalar operand.

The scalar unit can directly load/store into a SIMD register, treating each one as a small $8 \times 32$ b on-chip memory. This capability is used to allow eight scalar operands to be transferred between registers and memory in one cycle, and to allow 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 cycle.

SPERT instructions are stored in external memory and cached on chip. The instruction fetch unit manages a small instruction case that provides nearly 800 MB/s of instruction bandwidth. Typical applications are dominated by small loops and will experience high hit rates. Also, the large external memory bandwidth ensures that performance can never be more than halved as a result of instruction cache misses.

## III. 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. In this section we present detailed performance results for forward propagation and back-propagation training on SPERT.

These results have been obtained by careful analysis of assembly code routines. The results take into account 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 eight. This is not an important restriction, because dummy units can be inserted to pad smaller layers if necessary. For these results, weights are 16 b, inputs are 8 b, and activations are 8 b sigmoids looked up in a 64 K entry table. Most intermediate values are calculated to 24–32 b precision.

A single data structure is used to store the weight matrix for both forward propagation and back-propagation. Weights are stored as a two-dimensional array with all the weights corresponding to a particular neural unit in a column and the weights corresponding to a particular input in a row. Layer input and output values are arranged in linear arrays.

Forward propagation for a multilayered network is computed one layer at a time. The basic operation is a vector-matrix multiplication with a vector of input values used to form the inner-product with each column of the weight matrix. Within the layer, the output of eight units at a time are calculated—one unit per SIMD datapath. The first step is a vector-read of eight input values. One input is stored in each of the SIMD datapaths and subsequently broadcast to all the other datapaths. One by one each input value is broadcast and multiplied by a vector of eight weights read from memory; the results are accumulated, one per datapath. After all eight inputs

have been processed another set of inputs are read. The process is continued until the entire inner-product is computed for a set of eight units. Finally, the appropriate nonlinear function for each unit is computed with a table lookup operation using the scalar unit. The process is then repeated for another set of units until all units in the layer have been computed.

Fig. 4 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 350 MCPS. 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 unit helps SPERT attain high performance even with smaller nets. Nets must be smaller than $32 \times 32$ to drop below half peak performance.

The back-propagation training algorithm uses a forward propagation step to compute the error at the network outputs, followed by an error back-propagation step, and finally a weight update step. The forward propagation step is the standard method described above. The weight update step is simply a matter of accessing the weights and augmenting them with the appropriate error. The difficult part of back-propagation algorithm is the error back-propagation phase. This part is essentially a vector-matrix product between a vector of errors and the *transpose* of the weight matrix.

On SPERT, the weight matrix is not transposed in memory to better align with the SIMD datapaths; instead, inner-products are computed eight at a time, with each inner-product accumulated in eight partial sums, one per datapath. These 64 values are held in 8 of the SIMD general purpose registers. The inner loop of the back-propagation code reads in one group of 8 error terms, then reuses this value 8 times to multiply 8 vectors of 8 weights each, one vector per inner-product. The results are accumulated in the partial sums. Computation proceeds in these 8 by 8 blocks across the weight matrix until each of the 8 inner-products is available in the form of 8 partial sums that must be reduced to a single value. One SIMD operation can reduce the 8 elements to 4 which are then reduced to a single value using the scalar unit. This entire process is repeated for the next set of eight weight matrix rows.

Fig. 5 shows the training performance on a 3 layer feed-forward 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 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.

## IV. SPERT VLSI Implementation

The SPERT design is being implemented in MOSIS CMOS scalable design rules, with a target 50 MHz clock rate. The

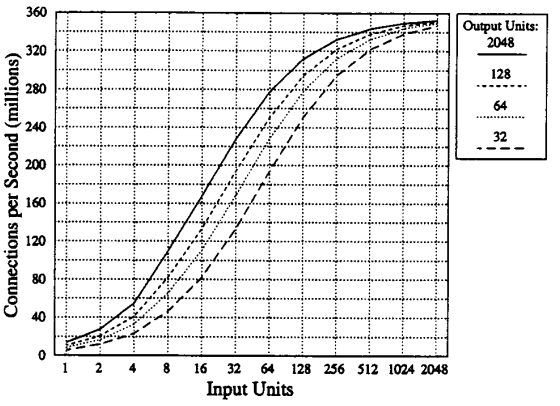


Fig. 4. SPERT forward propagation performance.

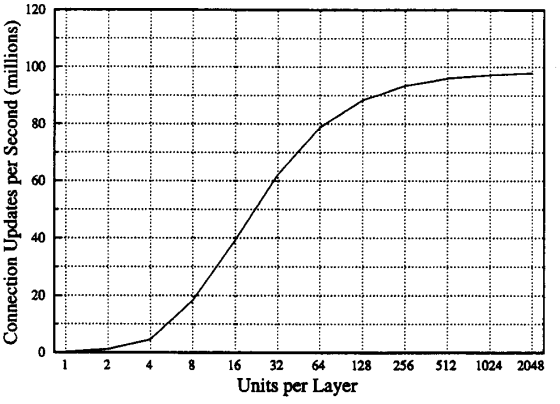


Fig. 5. SPERT training performance. Three layers, with equal number of units per layer.

design uses a mixture of full-custom cells for the datapaths and pads, and standard cells for control and other random logic.

A number of scaled SPERT components have been successfully fabricated and tested, including the critical multiplier and adder components. We have recently completed the design, layout, and verification of a test chip containing the datapath for the SIMD array. A microphotograph of the test chip is shown in Fig. 6. This datapath will be repeated eight times in the SPERT design. It measures $3{,}490 \times 9{,}132\ \lambda$ ($\lambda = 0.6$). Tests results for the fabricated chip indicate that the datapath is functional and operational above 50 MHz. Dynamic power consumption for the test datapath is 0.4 W at 50 MHz. Based on this measurement we expect the power consumption for SPERT to be around 5.0 W at 50 MHz. Fabrication of an entire neuro-microprocessor is planned for summer 1993.

## V. Related Work

Several related efforts are underway to construct programmable digital neurocomputers, most notably the CNAPS chip from Adaptive Solutions [2] and the MA-16 chip from Siemens [5].

Adaptive Solutions provides a SIMD array with 64 processing elements per chip, in a system with four chips on a

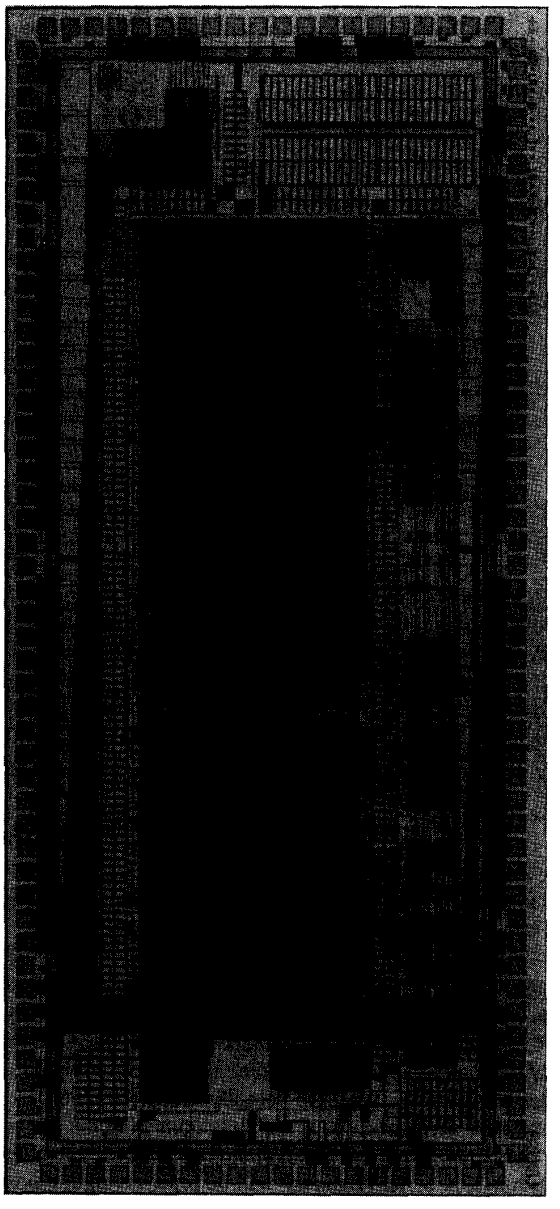

Fig. 6. Photomicrograph of SPERT test-chip.

board controlled by a common microcode sequencer. As with SPERT, processing elements are similar to general purpose DSP's with reduced precision multipliers. Unlike SPERT, this chip provides on-chip SRAM sufficient to hold 128 K 16 b 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 leverage the high density and low cost of commercial memory parts. This chip is designed to be most efficient for three general network formulae that are intended to summarize many connectionist computations. One realization will consist of a two-dimensional array containing 256 of these chips, and the resulting system will provide impressive raw peak throughput. This purely systolic approach, using deep pipelines and a specialized instruction set (at the chip level) will be appropriate for some network applications with little nonconnectionist code.

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 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.

## ACKNOWLEDGMENT

Brian Kingsbury has designed most of the fundamental circuit blocks that are being used in SPERT. Bertrand Irissou and James Beck have also been critical contributors to the VLSI design. Phil Kohn provided RAP software support for simulation studies. Thanks to John Lazzaro for sharing his expertise in chip design.

## REFERENCES

[1] K. Asanović, N. Morgan, and J. Wawrzynek, "Using simulations of reduced precision arithmetic to design a neuro-microprocessor," *J. VLSI Signal Processing*, 1993, to be published.

[2] D. Hammerstrom, "A VLSI architecture for high-performance, low-cost, on-chip learning," in *Proc. Int. Joint Conf. Neural Networks*, 1990, pp. II–537–543.

[3] N. Morgan and H. Bourlard, "Continuous speech recognition using multilayer perceptrons with hidden Markov models," in *Proc. IEEE Int. Conf. Acoustics, Speech, Signal Processing*, Albuquerque, NM, 1990, pp. 413–416.

[4] N. Morgan, J. Beck, P. Kohn, and J. Bilmes, "Neurocomputing on the RAP," in *Digital Parallel Implementations of Neural Networks*, K. W. Przytula and V. K. Prasanna, Eds. Englewood Cliffs, NJ: Prentice Hall, 1992.

[5] U. Ramacher, J. Blechter, W. Raab, J. Anlauf, N. Bruls, M. Hackmann, and M. Wesseling, "Design of a 1st generation neurocomputer," in *VLSI Design of Neural Networks*. New York: Kluwer Academic, 1991.

[6] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representation by error propagation," in *Parallel Distributed Processing. Exploration of the Microstructure of Cognition*, vol 1. Cambridge, MA: MIT Press, 1986.

[7] P. J. Werbos, "Beyond regression: New tools for prediction and analysis in the behavioral sciences," Ph.D. dissertation, Dept. of Applied Mathematics, Harvard University, Cambridge, MA, 1974.

**John Wawrzynek** received the M.S. degree in electrical engineering from the University of Illinois, Urbana/Champaign. He received the Ph.D. degree in 1987 from the California Institute of Technology where he worked under Carver Mead.

Currently he is an Assistant Professor of electrical engineering and computer science at the University of California, Berkeley, where he teaches courses in VLSI design and computer architecture. His current research interests include VLSI circuits and systems, neurocomputing, and parallel computer architectures.

Dr. Wawrzynek received the National Science Foundation Presidential Young Investigator Award.

**Krste Asanović** received the B.A. degree in electrical and information sciences tripos from Cambridge University in 1987. He is currently working towards the Ph.D. degree in computer science at the University of California, Berkeley.

From 1983 through 1989 he was with the GEC Hirst Research Centre, London. His research interests are in VLSI, massively parallel computer architectures, and object oriented programming languages.

**Nelson Morgan** (SM'86) received the Ph.D. degree in electrical engineering and computer science from the University of California, Berkeley.

He leads the Realization Group (a computer engineering department) at the International Computer Science Institute (ICSI), a nonprofit research laboratory closely associated with the Department of Electrical Engineering and Computer Sciences at the University of California, Berkeley. He is also an Adjunct Professor in the same department at the University of California. He is the author of Talking Chips, a book about speech synthesis with integrated circuits. His current interests include the design of algorithms, architectures, and systems for parallel signal processing and pattern recognition systems, particularly using connectionist paradigms.