A Supercomputer for Neural Computation

Krste Asanović, James Beck, Jerome Feldman, Nelson Morgan, and John Wawrzynek

Abstract— **The requirement to train large neural networks quickly has prompted the design of a new massively parallel supercomputer using custom VLSI. This design features 128 processing nodes, communicating over a mesh network connected directly to the processor chip. Studies show peak performance in the range of 160 billion arithmetic operations per second. This paper presents the case for custom hardware that combines neural network-specific features with a general programmable machine architecture, and briefly describes the design in progress.**

I. Introduction

The design and construction of a supercomputer specifically tailored for neural computation is a formidable task. We have undertaken this challenge with the twin goals of advancing the science of computer design while also producing a machine useful for connectionist research. To achieve these goals, we exploit several features of connectionist computation, as well as our own experience in systems development, to build the Connectionist Network Supercomputer (CNS-1). Recent work has shown the practicality of connectionist systems for a range of important problems, but it has also exposed the need for computational resources far exceeding those available to investigators in the field. The CNS-1 is designed to meet those needs, and will support research on problems orders of magnitude more complex than is possible with current machines.

Our earlier development of the Ring Array Processor (RAP) [7] has played a crucial role in our speech recognition research and is providing valuable insights for the design of the CNS-1 system. Experiments using the RAP show that limited precision fixed point arithmetic suffices for almost all algorithms of interest [2]. The high regularity of many connectionist tasks allows them to be efficiently mapped to distributed memory architectures and then executed using many parallel pipelines. Communication between processors is primarily involved with the broadcast of activation values, eliminating the read latency and memory coherency issues that plague many distributed parallelized applications. The design is further simplified by using the CNS-1 as a single-user, single-task attached processor, controlled by a host workstation.

Areas outside our work in connectionist speech research are also influencing the CNS-1 design. Applications being explored include language processing, auditory modeling, early and high level vision, and knowledge representation. Some of these applications require soft-real-time capabilities to handle live input such as speech or video. In other cases we are interested in connecting to analog interface systems being developed as part of our research [6].

Some of the most complex connectionist models include sparsely connected and sparsely activated networks. The semantic network is an extreme example of a sparse network, although it may also contain regions of dense connectivity. Existing connectionist coprocessors, including our RAP system, have not been designed or programmed to perform well on such models. The CNS-1 incorporates features to support sparse nets, and should encourage formulation of problems and models not previously attempted.

II. Architecture Development

Connectionist researchers have built computation engines using combinations of digital and analog elements for training as well as recognition tasks. The CNS-1 is an all-digital design based on custom fast VLSI chips interconnected to form a massively parallel processor. A fast processor, however, is not sufficient to warrant the effort of developing a complete system. If the *usable* performance of the system on everyday tasks (not just toy benchmarks) is not orders of magnitude greater than current workstations, then the project might be viewed as only an academic exercise.

Engineering a parallel processor system for high performance requires considering such issues as:

- Overall System Organization: number of inde-

The authors are with the International Computer Science Institute and the University of California at Berkeley.

pendent instruction streams, memory layout. control strategy.

- Node Architecture: instruction efficiency, application specific features, suitability to target technologies.
- Memory: hierarchy, size, bandwidth, latency.
- Internode Communication: topology, link bandwidth, protocol.
- I/O: locality, bandwidth, protection strategies, error recovery.
- Testing: bringup, debugging access, reliability, diagnostics.
- Supporting Hardware: power distribution, cooling, mechanical support, packaging techniques.
- Software: involved with all of the above items during the design phase, and then additionally with the user interface during the operation phase.

A pitfall common (but not unique) to parallel systems is Amdahl's Law. Paraphrased for our purposes, it states that the part of a task that could *not* be parallelized will be the bottleneck to overall performance. In parallel processor neurocomputers, the speedup is generally provided by the use of many fast circuits for the multiply-add operation. Certainly this is appropriate, since the common equations in use for learning and recall are well described as matrix operations, requiring many multiply-adds. The greatest possible speedup factor for a task, then, is defined as:

$$Speedup \leq 1/fraction\ not\ parallelized$$

In a 128 processor CNS-1, 1024 datapaths perform multiply-accumulates in parallel. If all the datapaths have to stop while a single unit executes 0.1% of the total operations in non-parallelized code, approximately *one half* of the elapsed time will be taken up in this bottleneck. This fact of life pervades the design process as we seek a balance between types of instructions, internode communication costs, computation and I/O, and, perhaps most importantly, ease of use for problems of interest.

The statistic most interesting to the neurocomputer researcher is problems solved per week rather than connections per second. Inadequate software means the desired algorithms will never be implemented satisfactorily, and the hardware design effort will be wasted.

The following subsections discuss these system design issues and outline the approach used in the CNS-1.

A. Instruction Balance

Most connectionist algorithms run best on systems that efficiently implement dot products and other matrix-oriented multiply-add operations. These operations generally do not require the range or resolution needed for traditional scientific computation. Single precision 32-bit floating point or even 16-bit fixed point representations appear to be sufficient. This paradigm of neurocomputers using parallel execution units with low or moderate precision dot product capability can be seen in other designs [4, 10].

However, to avoid the trap of Amdahl's Law, attention must be paid to those non-trivial manipulations of the inputs and outputs that occur with real algorithms. In our experience over the last three years with the RAP, we found that useful applications required such functionality. For example, inputs may need normalization, or preprocessing to compute derivatives. Similarly, outputs may need to be interpreted, normalized or even used in some entirely non-connectionist step, such as dynamic programming. Standard methods for dealing with these leftover chores include precomputing and saving values, performing part of the task on a different system, and making what's left as parallel as possible. Even more difficult are situations where conventional and connectionist computations are intertwined.

One measure of the success of a parallel system, as noted by Amdahl, is the ratio of time spent operating in parallel to time operating as a uniprocessor. In our case, the non-parallel computation time, t_{np}, for operations that are proportional to number of neural network input and output units begins to dominate the parallel execution time when

$$N \cdot t_{np} > N^2/P.$$

or

$$t_{np} > N/P.$$

This relationship holds for a network implemented on P processors with $O(N)$ inputs and outputs, $O(N^2)$ connections, and a single cycle multiply-add operation.

The message of this equation, which we confirmed during RAP performance testing, is that the parallel computation units need sufficient generality to be able to handle the non-dot-product operations. The processor for the CNS-1 includes that generality in the multiply-accumulate datapaths, providing support for parallel logical and arithmetic operations. These datapaths also include certain connectionist-specific features along with some support for emulated floating point arithmetic. The cost of adding this generality to the design is small relative to the improvement in overall system performance.

B. Internode Communication

Just as fast neurocomputer designs are based on low or moderate precision matrix arithmetic, efficient parallelization of these algorithms relies on the locality of computation. Using locality as a starting point, the designer trades off topology, bandwidth, latency and protocol in allocating the hardware resources. Quantifying these tradeoffs is complicated by the boundaries between different levels of the communication hierarchy.

One obvious hierarchy boundary useful for analyzing a system of parallel processors is between internode and intra-node communication. For the case of an array of VLSI parallel processors, transfers within a node may be over two orders of magnitude faster than the quickest internode message. To estimate the internode performance requirement, we can view a sub-net implemented on a single node in the same way a full network was viewed in the previous section.

For a worst-case analysis, we assume that each node has a complete representation of network state (that is, all of the activations, though only a piece of the weights). If a sub-net implemented on one node has $O(N^2)$ connections, $O(N)$ neural units and is implemented on P processors within the node, then the inter-chip communication time, t_{com}, begins to dominate the execution time when:

$$t_{com} > N/P.$$

As more processors are squeezed onto a chip, this restriction gets worse, especially since I/O capabilities are proportional to the perimeter of the chip, while on-chip computational capabilities are proportional to area. The restriction eases as more neural units are implemented on a node, but this does nothing to help overall system performance for a given problem size.

In the CNS-1, the network bandwidth is designed to comfortably meet this criteria at the node level for expected values of N/P. We have included some margin to allow for realistic traffic patterns in a mesh-connected network.

C. Input/Output

It is reasonable to assume that target neurocomputer applications are dominated by computation and communication within a tightly-coupled group of processing elements. However, as in the previous sections, the exceptions and bounds for such an assumption must be examined. Interaction with the host computer and disk may be orders of magnitude slower than any of the characteristic times for on-chip or between-chip operations. To give a worst-case example, random access of a pattern input on a disk can take tens of milliseconds, while arithmetic using this data can take tens of nanoseconds and be parallelized over hundred or thousands of processing elements. General-purpose network or message-handling routines that require operating system interaction can also easily take milliseconds.

Using the same analysis as in previous sections, it can be shown that since the I/O bandwidth is $O(N)$, the time for handling I/O, t_{io}, takes longer than the parallel execution time when

$$t_{io} > N/P$$

assuming the I/O is not parallelized. As before, larger values of P or smaller values of N make the requirement more stringent.

Of course, this worst case analysis ignores the advantages of memory hierarchies in dealing with the spatial and temporal locality in real problems [5]. For neurocomputing, these concepts come into play for any net architecture where multiple patterns are used as input to a net in a way that overlaps sequential net evaluations. Another strategy, if the local memory is large enough, is to store the entire input corpora locally and reuse it over multiple passes for training. Finally, since random disk access causes such a devastating performance penalty, algorithms must be organized to access larger blocks whenever possible. This enforces spatial locality on a disk block basis, even though the per-variable reads may be random within a block.

D. Software Requirements

The first concern for the neurocomputer researcher is flexibility. Of course, raw speed is important too, but if the machine cannot be coaxed into doing what is required, the performance advantage is nullified. The dual requirements for flexibility *and* speed tend to be contradictory. To be flexible, the user needs a general-purpose computing environment with a familiar language and a set of tools for building and debugging high level code. In order to obtain the maximum performance, the user needs to study hardware details and write intricate loops in assembler. In our experience, we have found it essential to support a wide range of users.

CNS-1 users will work within the general software development environment writing in C, C++ or Sather [9]. Using calls to hand coded library routines written in assembly language, they will access optimized functions such as matrix-vector multiplies. Since all of the code is run on the target neurocomputer, there is no overhead for remote procedure calls between host and neurocomputing engine. For the CNS-1 system, we have developed high level neurocomputing simulation tools to allow the

sophisticated user to develop code without access to the hardware.

III. CNS-1 System Overview

The CNS-1 is a digital multiprocessor system designed for the moderate precision fixed point operations used extensively in connectionist network calculations. We are building a custom VLSI digital processor employing an on-chip vector coprocessor unit tailored for neural network calculations and controlled by a RISC scalar CPU. This RISC processor, called Torrent-1, is compatible with the industry standard MIPS-II instruction set architecture.

One processor and associated commercial RAM comprise a node, which is connected in a mesh topology with other nodes to establish a MIMD (Multiple Instruction, Multiple Data) array. Each node contains a private memory space and communicates with others through a simple message passing protocol. The CNS-1 will be built with 128 processing nodes giving a maximum computational capacity of 160 billion integer operations per second and a total of 2GB of storage. The design is scalable to 512 processing nodes for up to 640 MegaOps and 8GB of RAM. One edge of the communications mesh is reserved for attaching various I/O devices, which connect via a custom network adaptor chip, called Hydrant. The CNS-1 operates as a compute server, and one I/O port is reserved for connecting to a host workstation.

A. Torrent Processor Chip

The CNS-1 processor chip, Torrent-1, includes a MIPS-compliant CPU with a vector coprocessor, on-chip caches supported by a fast external memory interface, and a high performance network connection. The target clock rate for Torrent is 80MHz. By including a complete RISC CPU (rather than simply a specialized controller for the vector units), routine scalar tasks can be handled local to the parallel units. Additionally, the choice of a standard instruction set architecture means that widely available tools can be incorporated in our support software.

The vector coprocessor on Torrent accelerates neural computation by executing up to 16 moderate precision fixed point operations per cycle. The MIPS CPU provides general scalar processing and supports the vector coprocessor through address generation, loop control and by providing scalar operands.

To supply the high memory bandwidth needed to keep the vector unit busy, a vector memory pipeline controller is included on-chip. Fast SRAM is used to provide an aggregate memory bandwidth of over 1.2 GB/s with 16MB of SRAM per Torrent[1]. The instruction fetch bandwidth is supported by an on-chip instruction cache.

Torrent also includes a hardware router to handle communications in the CNS-1 mesh. The communications path is tightly integrated with both the scalar and vector processors, and supports a low overhead transfer protocol based on active messages [3]. The appeal of active messages for CNS-1 is that a message arriving at its destination triggers execution of a local event handler, allowing very fast response for short messages. The ability to customize these handlers will be important for neural nets with sparse interconnections and activations, where the balance between computation and communication is particularly critical [8].

B. Hydrant I/O Chip

Communication between the networked Torrent processors and the outside world takes place through a custom network adapter chip, named Hydrant. Attached along one edge of the communication mesh, Hydrants convert the high speed CNS-1 protocols to a more general parallel interface. This interface is then connected to a field programmable gate array to allow customizing the signals for particular interfaces. The host and other devices will all connect to CNS-1 through Hydrant nodes.

IV. Software

Making the CNS-1 into a usable system requires a major effort in software development. During the design phase of the project, dozens of programs have been modified or written from scratch to support the VLSI effort.

Using the GNU suite of tools as a starting point, a programmer's environment has been constructed to support running code on software simulators (now) or the working hardware (later). A parallel effort is underway to provide support for testing hardware and debugging code.

A long term emphasis at ICSI and UCB has been the development of connectionist simulators. One simulator, CLONES, has been used successfully on the RAP for training large backpropagation networks used in speech recognition research. Our newest generation simulator, BoB (for Boxes of Boxes), builds on this experience and includes features added specifically for the CNS-1.

At the highest level, BoB allows users to build data flow maps using BOX and PORT classes. The separation of computation and communication simplifies the design step and allows lightweight ab-

[1] An alternate implementation would be to use synchronous DRAM, or SDRAM, with approximately the same performance.

stract classes to be defined. BoB is particularly targeted for MIMD hardware, such as CNS-1, and will efficiently support non-connectionist calculations.

The BoB library includes a set of classes that support general connectionist and speech-oriented databases which may be larger than a single disk drive. Such a logical database object can hide the fact that it consists of many files containing data for different regions of the full database.

V. Status

The Torrent architecture is fully specified for the computation units and partially specified for the network controller. The first implementation of Torrent, called T0, is in the final stages of design and should be operational in the fall of 1994. T0 does not include an integral network controller, and is targeted for single node implementations.

Most of the T0 design will be reused for the CNS-1 Torrent processor, called T1. The network interface design is underway, and will be incorporated in both T1 and the Hydrant I/O chip.

Much of the software framework is in place and work is underway to map applications to the CNS-1.

VI. Conclusions.

Experience with neurocomputer design and use has shown us that many of the standard requirements for good computer system design are still important, even if we are simulating neural computation that is significantly different from most common computer applications. In particular, we must do a system-oriented design, paying significant attention to the execution of scalar and other non-connectionist operations, both intra- and inter-chip communication costs, I/O between the host/disk subsystems and the neurocomputing engine, software, and diagnostic capabilities. Careful attention to these details can result in a system that provides sustained performance within a small factor of the peak for a wide range of relevant neurocomputing problems, and flexible programming capabilities for users.

VII. Acknowledgements

In addition to the authors, the primary CNS-1 design team members are Tim Callahan, Bertrand Irissou, Dave Johnson, Brian Kingsbury, Phil Kohn, John Lazzaro, Thomas Schwair, and David Stoutamire. The National Science Foundation provided financial support through Grants MIP-8922354 and MIP-9311980, and with Graduate Fellowships. John Wawrzynek received support from the National Science Foundation through the Presidential Young Investigator (PYI) award, MIP-8958568. Primary funding for the project is from the ONR, URI N00014-92-J-1617, ARPA contract N0001493-C0249, and the International Computer Science Institute.

References

[1] K. Asanović, J. Beck, T. Callahan, J. Feldman, B. Irissou, B. Kingsbury, P. Kohn, J. Lazzaro, N. Morgan, D. Stoutamire, and J. Wawrzynek, CNS-1 Architecture Specification - A Connectionist Network Supercomputer, *ICSI Technical Report,* TR-93-021, April 1993.

[2] K. Asanović and N. 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.

[3] T. von Eicken, D. E. Culler, S. C. Goldstein and K. E. Schauser, Active Messages: a Mechanism for Integrated Communication and Computation, *Proc. Tenth International Symposium on Computer Architecture*, May 1992.

[4] D. 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.

[5] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach,* Morgan Kaufmann, San Mateo, 1990.

[6] J. Lazzaro, J. Wawrzynek, M. Mahowald, M. Sivilotti and D. Gillespie, Silicon Auditory Processor as Computer Peripherals, *IEEE Journal on Neural Networks,* May 1993.

[7] N. Morgan, J. Beck, P. Kohn, J. Bilmes, E. Allman and J. Beer, The Ring Array Processor (RAP): A Multiprocessing Peripheral for Connectionist Applications, *Journal of Parallel and Distributed Processing*, Special Issue on Neural Networks, v14, pp.248-259, 1992.

[8] S. Mueller and B. Gomes, A Performance Analysis of CNS-1 on Sparse Connectionist Networks, *ICSI Technical Report,* TR-94-009, February 1994.

[9] S. Omohundro, The Sather Programming Language, *Dr. Dobb's Journal*, Volume 18, Issue 11, p. 42, October 1993.

[10] 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.