Designing a Connectionist Network Supercomputer[1]

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

University of California and the
International Computer Science Institute
Berkeley, California

September 10, 1993

Abstract

This paper describes an effort at UC Berkeley and the International Computer Science Institute to develop a super-computer for artificial neural network applications. Our perspective has been strongly influenced by earlier experiences with the contruction and use of a simpler machine. In particular, we have observed Amdahl's Law in action in our designs and those of others. These observations inspire attention to many factors beyond fast multiply-accumulate arithmetic. We describe a number of these factors, along with rough expressions for their influence, and then give the applications targets, machine goals, and the system architecture for the machine we are currently designing.

1 Introduction

The goals of the Connectionist Network Supercomputer-1 (CNS-1) project are to produce super-computer performance and flexible software for connectionist computation at moderate cost. In a university-based research environment, there are two reasons for undertaking an ambitious development project: either the result is needed or one expects the effort to advance science and technology. Both motivations are equally important for CNS-1. Recent work in our lab and elsewhere has shown the practicality of connectionist systems for a range of important problems, but has also revealed needs for computational resources far exceeding those available to investigators in the field. Our own earlier development of the Ring Array Processor (RAP) [8] attached processor has played a crucial role in our research and is proving valuable to other groups who have acquired the system. CNS-1 will supply orders of magnitude more capability in a form that we already know how to exploit.

There are several features of connectionist computation that are exploited in the CNS-1 design. Experiments show that limited precision fixed point arithmetic suffices for almost all algorithms of interest[1]. Many problems are highly regular and are well suited to parallel and pipelined execution. Connectionist networks are "embarassingly" parallel and map nicely to distributed memory machines. Communication is usually multi-cast and normally values only are sent, eliminating the read latency that plagues most distributed computing. Another major source of simplification in the CNS-1 design is that the machine will be used as a single-user, single-task attached processor; this eliminates many of the most complex hardware and software problems in parallel computing and fits our application requirements.

Our experience with connectionist speech research on the RAP permits a fairly concrete consideration of these issues. However, we have also considered a number of other

[1] To appear in the *International Journal of Neural Systems.*

appropriate applications such as language processing, auditory modeling, early and high level vision, and knowledge representation. In each of these cases we have good reasons to expect that we can benefit from an architecture that is slanted toward moderate-precision representation, both in raw performance and in price/performance.

A number of these applications require soft-real-time capabilities to handle real-world input such as speech, and many of them could benefit from system implementations using a relatively small number of nodes. In some cases we are interested in interfacing with specific analog interface systems that are part of our research [7]. These requirements provide a strong impetus for users to want a significant say in the design of the computing system.

In addition to the specific connectionist applications of CNS-1, we are very interested in the potential of CNS-1 as a general purpose connectionist accelerator. The connectionist model of computation differs in many ways from conventional models and we exploit these differences in the design. But the mismatch has meant that simulations of connectionist models have been sufficiently clumsy and inefficient to constitute a major barrier to advanced research and development. In some cases, such as speech recognition, we already have clear ideas on how to exploit the capabilities of CNS-1. There are other domains, such as early vision and auditory modeling, where the outlines of promising approaches are understood but the details remain to be worked out. Even more exciting to us are the prospects that a connectionist super-computer will encourage the formulation of problems and models that were not previously taken seriously because of impracticality.

The architectural keystone to supporting the full range of connectionist models is the treatment of sparsely connected and sparsely activated networks. All existing connectionist co-processors, including our own RAP system, have their hardware and software oriented towards densely connected feed forward networks. CNS-1 will still be most efficient for such nets, but considerable design effort has gone into support for general networks. A paradigm case has been human-like semantic memory where the activation of one concept automatically activates related concepts. Such a capability is an essential part of higher level vision and language understanding and could greatly extend the applications of CNS-1 in traditional AI domains. The semantic network is an extreme case of sparseness; many practical problems consist of a mixture of dense and sparse connections and CNS-1 should retain most of its performance benefits for such tasks.

In the next section, we will discuss some of the design requirements for a useful neurocmputing system (useful in the sense of accelerating research into neural network applications such as those alluded to above).

2 Requirements of Neurocomputers

In the early days of the current resurgence of interest in neural networks, it was widely thought that the use of analog VLSI implementations were both necessary and sufficient for neural hardware research. It is now clear that neither of these perspectives is strictly correct.

A number of connectionist researchers have built powerful computational engines with entirely digital elements, entirely analog designs, and sometimes a hybrid of the two. An example of the last category was the NET32K chip from AT&T[4]. This chip used bitwise multiplications, digital weight storage, and digital I/O, while performing addition by summing currents. Most researchers now understand that the design choice between analog and digital should be dictated by the research goals (e.g., biological modeling vs. programmable research tool) and by the system requirements for a specific application.

For almost any design goal, a fast computational chip is not enough. The entire system must be engineered, considering such issues as test and debug, memory bandwidth and latency, communication between processors, I/O, and software. For the last category, even in the case where programmability is not a design goal, there must at least be an interface to a programmable host's software. Without these considerations, the hardware may never be fully debugged; even if it works, the observable performance of the specialized artificial neural hardware could be abysmal (potentially lower than workstation performance for real daily use).

As a community we are beginning to learn the lessons that computer system developers learned long ago, as typified by Amdahl's Law. This fundamental law is currently stated in a number of ways, but the basic idea is that the improvement of overall system performance due to the speeding up of one part of the system (e.g. faster computation via parallelism) is limited by the fraction of the job that is not speeded up (e.g. serial code). For instance, if 90% of a task is speeded up infinitely, the overall task is only done ten times as quickly.

In the neurocomputing context, the speed-up is generally provided by the use of many fast circuits for the multiply-accumulate operation. Whether this is done by continuous-time analog circuits, pulse-mode arithmetic, or entirely digital means, such fast arithmetic is fairly universal as the primary model of neurocomputing acceleration. Certainly this is appropriate, since all of the common equations in use for learning and recall are well described as matrix operations, requiring many multiply-accumulate operations.

However, as with conventional computing systems, Amdahl's Law still holds for neurocomputers. Stated in the context of this paradigm, Amdahl's Law (for Neurocomputers) would be:

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

Thus, a connectionist capability of a billion connections per second coupled with a conventional capability of 10 million arithmetic or logical operations per second presupposes a ratio of 100 to 1 between neural and non-neural operations (to avoid the non-neural sections being the limitation on performance).

Of course, such a statement vastly oversimplifies the issues, but in general the concept should hold for real systems. In real systems, though, balance is not only required between classes of instructions, but also between computation and I/O, interprocessor communication costs, and, perhaps most importantly, ease of use for problems of interest. The statistic most interesting to the neurocomputer user is probably problems solved per week rather than connections per second. If the software is insufficient to the task, then the desired algorithms will never be translated satisfactorily to the machine, and the hardware will be useless. This is not idle speculation; in fact the majority of all parallel research machines have never been used outside of machine validation and toy applications.

The remainder of this section will discuss each of the system requirements mentioned above, and will describe some of what was learned about balancing systems during the design of a neurocomputer that we have been using. These considerations are shaping the design of the CNS-1.

2.1 Instruction Balance

Connectionist algorithms require systems that can efficiently implement dot products and other matrix-oriented multiply-accumulate operations. It is also true in general that these operations do not typically require the kind of range or resolution that is required for general scientific computation; single-precision 32-bit floating point or even 16-bit fixed point representations appear to be sufficient [1]. Thus, neurocomputing systems nearly always consist of a parallel system with low or moderate precision dot-product capability [5, 10].

However, it is often true that non-trivial manipulations of the inputs and outputs of the universal matrix operations must take place in real algorithms. In our experience over the last three years with the Ring Array Processor (RAP) [8] we have often found that applications required such functionality. For example, inputs often need to be normalized, or preprocessed to compute first or second derivatives. These cannot always be precomputed and stored, as they can take up a prohibitive amount of disk space for real applications. Similarly, outputs may need to be compared to values, interpreted, normalized, or even used in some entirely non-connectionist framework, such as dynamic programming. In the general case, if we say that a network has $O(N^2)$ connections, $O(N)$ inputs and outputs, and is implemented on P processors, then the computation is dominated by the non-connectionist component if this aspect takes t_{nc} steps per input or output unit, where

$$t_{nc} \; > \; N/P.$$

These considerations are not theoretical in origin, but are posthoc observations from having been surprised at the length of time required to execute real algorithms on the RAP. However, the consequences of this inefficiency were not as bad as they could have been with an overly-specialized design, because the RAP was composed of general-purpose computational components (commercial DSP chips), so that many of the non-dot-product operations could also be parallelized This has suggested to us that in newer designs we must retain some capability for more general computation. In the processor design for the CNS-1 the multiply-accumulate datapaths have been generalized somewhat further to contain general logical and arithmetic instructions and some specific features that will speed software emulations of floating point arithmetic. These modifications have actually added very little area to the datapath, but have broadened its capabilities considerably.

In general, then, we feel that some general capability for non-connectionist instruction execution must be retained at the node level in order to maximize efficiency. Of course, this choice is not made without cost.

2.2 Interprocessor Communication

Much as the fast neurocomputer realization of connectionist computation is predicated on the dominance of low or moderate precision matrix arithmetic, fast parallelization of these and other similar algorithms depends on the locality of most computation.

It is more difficult to quantify the tradeoffs in this category than it was in the previous section, as the costs depend on the nature of the boundaries between different levels of the communication hierarchy. For instance, in principle a systolic architecture has little or no communication cost (other than pipe-filling latencies, assuming that communication between systolic elements occurs concurrently with computation). On the other hand, in real applications if the pattern of data manipulation is not a perfect fit to the physical structure, inefficiencies are inevitable. Again the requirements of mixed computation will have similar consequences for computation. Furthermore, it is generally possible to have much higher interprocessor bandwidth on-chip than it is off-chip. Viewing a sub-net that is implemented on a single chip in the same way as a full network was viewed in the previous section, we can come to similar conclusions for at least the case of inter-chip communications. For a worst-case analysis, we assume that each chip must have a complete representation of network state (that is, all of the activations, though only a piece of the weights). If a network has $O(N^2)$ connections and $O(N)$ neural units and is implemented on P processors spread over the chips, then the execution time is dominated by the inter-chip communication if this aspect takes t_{com} steps per unit, where

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

Clearly as more and more processors are squeezed on to a chip this problem gets worse, especially since input-output capabilities are in some sense related to the perimeter of the chip, while on-chip computational capabilities are related to area and density. This is a difficult case, since one may not need to pass all activation values to all chips. On the other hand, sparsely connected networks by definition have a worse connections/units ratio.

Therefore, as with general-purpose parallel computing, high off-chip communication bandwidth is necessary for efficient neurocomputing. In the Torrent-based CNS-1, the network bandwidth is somewhat higher than the above formulation would suggest (the Torrent can read in several activations per cycle, making the factor t_{com} above less than one), since we wished to allow for realistic traffic in a network that was more general than a simple ring.

2.3 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 with the previous sections, the exceptions and bounds for such an assumption must be examined. Interaction with the host computer and/or disk is often orders of magnitude slower than any of the characteristic times for on-chip or even 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 only a few 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. Host interaction latencies can be lower, but a full microprocessor-neurocomputer handshake over a standard commercial bus can still take microseconds. Again, this is not merely a fanciful horror story; a number of research applications on real neurocomputers have been I/O bound, in some cases running a thousand times slower than the theoretical peak rates.

Using the same analysis as in previous sections, it can be easily shown that a time constant of $t_{io} \;=\; N/P$ cycles is the boundary for an I/O bottleneck to a neurocomputer. As before, larger values of P or smaller values of N make the requirement more stringent. Put another way, the communication from the host or disk must be roughly as fast as the inter-chip data transfer.

Of course this was a worst case analysis; the reason that memory hierarchies are successful is that there is some spatial and temporal locality in real problems [6]. These concepts are still relevant for neurocomputing. For instance, a common case of temporally local access is that of convolutional network inputs, or other architectures in which multiple patterns are used as input to a net in a way that overlaps for sequential net evaluations. In other words, inputs are commonly re-used even within a single pass through a network. Also, if the local memories are large enough, entire input corpora can be stored locally and re-used over multiple passes for training. Finally, since random disk access causes such a devastating time penalty, algorithms must be organized to access larger blocks whenever possible. This enforces spatial locality on a block basis (since reading a large sequential chunk from disk is much more efficient than numerous small chunks), even though the per-variable reads are random within a block. For instance, even with random pattern presentation schemes, a large but incomplete chunk of the input data may be read in to the system memory and then probed randomly.

The I/O system for the CNS-1 design presents is a challenge, as the largest system supported will have 8000 datapaths operating at 125 MHz. This necessitates a significant effort for the design of a high-performance I/O subsystem, but certainly also forces the user to develop algorithms to maximally re-use data. However, the design will also incorporate large amounts of local memory (32 GB for the largest system) so that many input corpora will fit on the machine for reuse, particularly in multiple passes for learning.

2.4 Software Requirements

Flexibility for research is the most major single concern of the neurocomputer user. Of course, raw speed is important too, but if the machine cannot be coaxed into doing what is required, the theoretical speed is useless. In practice the needs for flexibility and speed are often contradictory requirements. The user needs a general-purpose computing environment in which he can use a general-purpose programming language. Often one must port existing serial code. Point-and-click neurally-oriented interfaces can simplify demonstrations or toy problems, but typically are not sufficient for the general kind of programming that users require for research in a complete application. On the other hand, simply running serial code through a compiler will not take advantage of the special capabilities of the neurocomputer.

In our systems we have found it essential to use general computing environments in combination with specialized hand-coded library routines. Users program the RAP in C++ or C (or, more recently, in an ICSI-developed object-oriented language called Sather). In the midst of these codes are calls to library routines that are hand coded in assembly language. These functions do such common operations as matrix-vector multiplies. Since all of the code (including the non-library sections) is typically implemented on the neurocomputer, there is no overhead for remote procedure calls between host and neurocomputing engine.

Similar libraries are now being developed for the CNS-1 system. Additionally, high level neurocomputing simulation environments are also being developed, but in this case as well we preserve the capability for knowledgeable users to generate their own code.

3 System Targets

3.1 Applications Focus

We have developed a number of application targets for the CNS-1. These range from extensions of tasks we have already run on parallel computers through bona fide applications that have never been implemented on fast hardware. In addition, we have developed an abstract target application that is representative of classes of problems that we have previously been unable to handle. Together these applications imply a number of system goals for the CNS-1.

Within our research group we have been doing large computations for connectionist speech research. In particular, we have been training large Multi-Layer Perceptrons (MLPs) to estimate phonetic probabilities for continuous speech recognition [2]. These techniques are currently quite competitive with the best classical statistical approaches.

Training our networks for the standard DARPA Resource Management task requires on the order of 10^{14} arithmetic operations. Depending on the details of the run, this costs a few days of computing on our current parallel computer, the RAP. New tasks that we are attacking require significantly more computing due to larger databases and larger, more complex networks that are required to handle more complex recognition tasks. Additionally, we need to iterate on these training runs more often in order to test out new robust feature extraction methods. We estimate that we need computational power that is 2 to 3 orders of magnitude larger than we have with the RAP. This translates to a system that is 4 to 5 orders of magnitude more powerful than a Sparc 2 workstation, or perhaps 3 to 4 orders of magnitude more powerful than a 1995-vintage workstation.

In addition to raw network and training set size, we expect to require a more diversified instruction mix. In particular, there are many good reasons to tightly couple dynamic programming steps with the network evaluations. This suggests that dynamic programming and pointer bookkeeping must be possible within CNS-1.

Finally, as the speech networks (currently over a million connections) get larger, we tend more towards networks that are not fully connected. However, in our experience this sparseness is generally not in the form of random connection vacancies, but rather in the form of fully-connected subnets that are glued together in application-specific ways. Such "chunky" networks must be supported in the CNS-1 machine for the speech recognition research application.

While our greatest experience with parallelizing large network applications is with speech, we have examined a number of other tasks during the design of the CNS-1. These tasks include language processing, auditory modeling, early vision, high level vision, and large conceptual knowledge representation studies. We are currently studying possible mappings of these tasks to the architecture.

In addition to these concrete tasks, it is important to consider a more abstract problem that may clarify some of the costs and tradeoffs resulting from design decisions. To that end, we have defined the following problem description:

> *Evaluate a network with a million units and an average of a thousand connections per unit for a total of a billion connections. This should be done 100 times per second.*

This description is deceptively simple, and does not actually specify the distribution of connections. However, this problem allows us to examine the relationship between this connectivity and the performance for alternate design choices. Since we do not yet know the relation between any form of this problem and the more concrete tasks mentioned earlier, we want to be sure that less common forms of this task are not impossibly inefficient on the CNS-1. However, we suspect that most tasks of interest will have some locality or "clumpiness" to the connection matrix, as we have observed such behavior on most real problems.

3.2 Machine Goals

Given the above target tasks and perspectives, we can summarize the CNS-1 performance goals as follows:

- Connectionist compute power. The minimum configuration should evaluate 100 billion connections per second, which therefore requires a weight-reading bandwidth of 200 billion bytes/second for 2-byte weights.

- Learning capability. Back-propagation learning including weight updates should be accomplished in at worst 1/5 of the evaluation rate.

- Communication capability. For the large abstract problem, we will require a broadcast of 10^6 activations 100 times per second. Each node must then read 10^8 B/sec from the network (the inputs from all units), but write out a much smaller amount (only the outputs from the units represented locally).

- Storage. Two gigabytes are needed to hold a billion 2-byte connections. Including activation tables and pointers for sparse networks, another 2 GB are required. This 4 GB is really a minimum, and larger systems will be required to simultaneously hold large data sets. We expect our initial system to be limited to the 4 GB figure since our current tasks actually use smaller networks (though they have very large data sets, e.g. 1–2 GB), and the large abstract problem does not at this time have a corresponding large input data set.

- Sparseness. A network with arbitrary sparseness should still run on CNS-1, although a network composed of fully connected subnets will be more efficient. The performance degradation with increasingly random sparseness should be gentle, so that a fully random interconnection pattern should be evaluated at a speed that is no worse than 1% of peak, and normally sparse networks should work at greater than 10% efficiency.

- Shared Weights. Many of the tasks described above incorporate shared or tied weights. While in some cases these weights have been tied to save storage, shared weights are a more useful way to enforce properties such as shift-invariance, as well as to reduce parameters for smooth estimators. Shared weight evaluation and learning must be supported.

- Non-connectionist processing. Since many applications require integrating operations that are not fixed point matrix-vector operations, the CNS-1 node must be able to perform general scalar operations at a rate that is only moderately reduced from the peak rates of the more common operations. Some common non-multiply-accumulate vector operations should also be supported.

- Numerical representations. We expect to take advantage of the fact that the variable we process, store, and communicate are short. For most purposes, 1-byte numbers are sufficient to represent unit activations and 2-byte numbers are sufficient to represent connection weights. These common choices generate far-reaching consequences in the reduction of resources to achieve the computational and communication goals described above.

- Floating point support. It is important to implement multiple precision and floating point representations. These will be important for the development stages of fixed point algorithms, when scaling or range properties of the variables may not be well understood. Additionally, the final form of coded algorithms may wish to use some floating point or double precision fixed point.

- Coding capabilities. CNS-1 is intended as a fully programmable computer, and is not just a fixed function back-propagation machine. Software must be developed to give CNS-1 the "look and feel" of a more general computer, if it is to be anything beyond an academic curiosity. The use of carefully hand-coded library routines will provide efficient operation for the common functions that we anticipate. However, another design goal must be graceful performance degradation for the general (non-parallel) operations included in the programmer's code. Provision for clean mechanisms of expression for these programming modes is another key requirement for the system.

- Soft real time capabilities. An important aspect of the machine's intended function is the ability to interface interactively with analog sensors and actuators. For instance, real-time streams of speech and video images must be accepted by the machine and processed roughly within the appropriate frame times. There should be a simple high-performance interface mechanism that can be used to connect CNS-1 to a variety of target external devices.

4 CNS-1 System Overview

4.1 Architecture

The connectionist network supercomputer (CNS-1) is a multiprocessing system designed for connectionist network calculations and other moderate precision integer calculations. The architecture of the CNS-1 system is similar to that of other massively parallel computers; major differences arise in the architectural details of the processing nodes and the communication mechanisms. The hardware is illustrated in Figure 1. Custom VLSI digital processing nodes are tailored for neural-network calculations. Processing nodes are connected in a mesh topology and operate independently in a MIMD style. Each node contains a private memory space and communicates with others through a simple message passing scheme. The initial implementation will be built with 128 processing nodes giving a maximum computational capacity of 256 billion integer operations per second and a total of 4GB of storage. The design is scalable to 1024 processing nodes for a total of up to two TeraOps and 32GB of RAM. One edge of the communications mesh is reserved for attaching I/O devices, allowing up to 8GB/s of I/O bandwidth. The CNS-1 array is attached as a compute server to a host workstation.

The processor, named Torrent, includes a MIPS CPU with a vector coprocessor running at 125MHz, an external memory interface with on-chip instruction and data caches, and a network interface. The inclusion of a full scalar CPU (as opposed to some more specialized controller for the vector units) means that routine scalar tasks will be supported so that they do not dominate computation in complete tasks (as noted in the Amdahl's Law discussions earlier). Additionally, the choice of a standard Instruction Set Architecture (ISA) means that many commercial tools can be incorporated in our software support.

The vector coprocessor accelerates neural computation. It contains multiple moderate precision fixed-point datapaths that complete up to 16 operations per cycle, yielding up

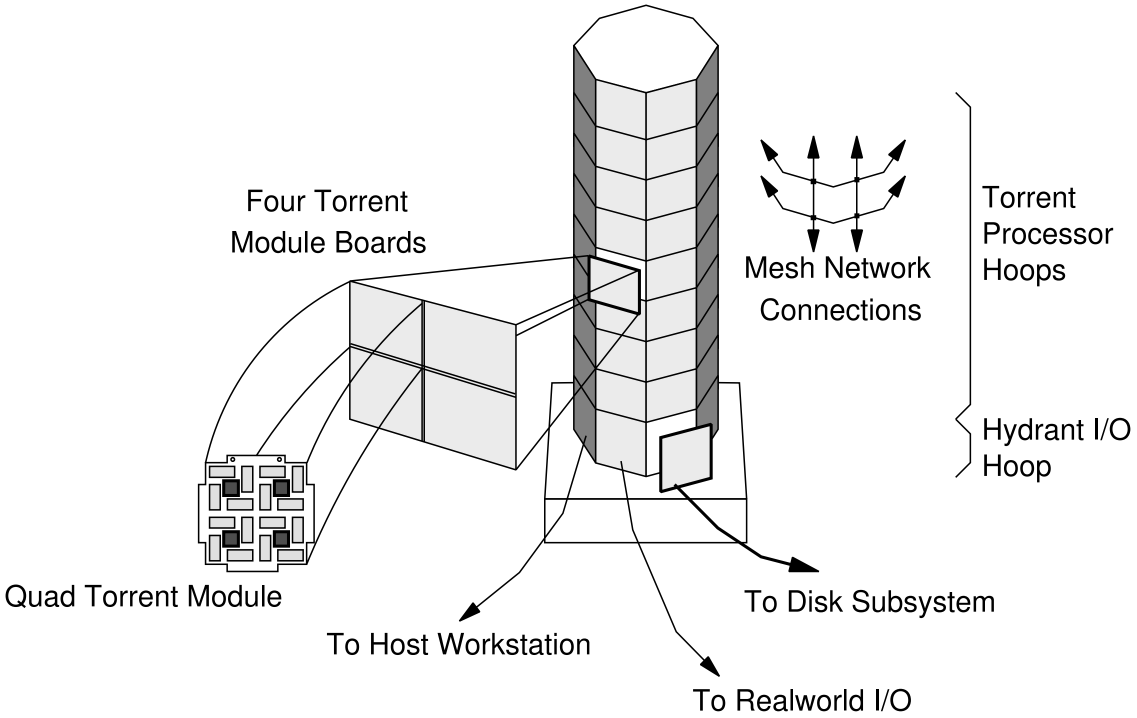

Figure 1: CNS-1 Hardware.

to two billion operations per second. Torrent includes a 32b RISC scalar unit; we find it important to provide tightly coupled scalar processing since many applications have a significant non-neural component. The MIPS CPU provides general scalar processing and supports the vector coprocessor by providing scalar operands, address generation, and loop control.

To provide the high memory bandwidth needed to feed the vector unit, much care is being put into the memory interface. A fast form of DRAM (either Rambus or Synchronous DRAM) will be used to provide an aggregate memory bandwidth of over 1GB/s and roughly 32MB of DRAM per Torrent. An on-chip data cache is included to reduce average memory access latencies, but may be bypassed when accessing vector data to avoid cache pollution. The instruction bandwidth is provided by an on-chip instruction cache.

Torrent includes an on-chip hardware router to handle communications in the CNS-1 mesh. A related and important feature of Torrent is a fast processor-network interface. CNS-1 has some unique requirements for data communication that are not well met by existing massively-parallel computer systems. Effective execution of neural network architectures with sparse interconnections and activations requires efficient communication of small messages. The communication and management of continuous sensory data streams require rapid processor response to asynchronous external events. Torrent has a fast and simple network interface with messages sent and received directly from processor registers. The active message model [11] is directly supported, where an arriving message triggers execution of a local event handler.

System design is simplified because each processing node comprises only a single chip processor and DRAM. No external logic is necessary to complete a node, increasing reliability and minimizing the board area required per processor.

The host and other devices will connect to the processors of the CNS-1 through custom VLSI I/O nodes attached to an edge of the communications mesh. The I/O node, named Hydrant, includes the same network router as Torrent together with a set of message send and receive buffers that can be accessed over a conventional parallel interface. Additional external components can be used to customize a Hydrant to provide a wide variety of I/O interfaces, including standard peripheral interfaces, such as HIPPI and SCSI, as well as custom interfaces to sensors and actuators. In early versions of the machine the host will provide access to disk storage. Later improvements will include direct interfaces to fast mass-storage subsystems without host intervention.

A mesh topology is used for the data network connecting Torrent and Hydrant nodes. A network link between neighboring nodes consists of 8b of data and an acknowledgment wire in each direction. The data network is synchronous with a clock rate of 125MHz, giving a peak data throughput of 125MB/s in each direction per link. The largest CNS-1 system with 1024 Torrent nodes has a bisection bandwidth of 8GB/s, or 8MB/s per processor.

A bit-serial diagnostic network is provided for hardware diagnostics, bootstrapping, and run-time monitoring. This diagnostic network is a high-performance extension of the industry standard JTAG and is controlled by the host workstation.

A preliminary form of the Torrent chip called T0, which lacks the network and DRAM interfaces, is currently (July 1993) near completion.

4.2 CNS-1 Software

A considerable amount of software development is required to make CNS-1 a usable system. At the lowest level we require diagnostic routines that will be used to detect hardware failures in the machine. These routines are written for the host system and take over CNS-1 for the duration of the diagnostic tests.

Users start their applications from the host by invoking a `cnsserver` and giving it the name of a CNS-1 executable. The `cnsserver` process first resets, then performs a bootstrap sequence which ends by downloading the executable to all processing nodes in the array. The application is started by a message sent from the `cnsserver` at this point. `cnsserver` then enters a monitoring loop, repeatedly scanning out error flags and profiling information across the diagnostic network. In the early CNS-1 system, the host is also used for file I/O, and so `cnsserver` may be interrupted with system I/O requests from the array. Finally, the application will send an exit message back to `cnsserver` which will then clean up on the host and release the machine for the next user.

A number of languages will be supported by CNS-1. A Torrent assembler will be an important tool in the development of the optimized libraries. Both C and C++ will be made available by a port of the GNU `gcc` compiler to the Torrent processor. This C compiler will then be used as the target for a port of Sather [9]. Parallel language support for distributed memory machines is still very much a research issue. We expect to use experience gained in a pSather [3] port to the CM-5 to port pSather to CNS-1. The CNS-1 hardware is simple, fast, and flexible, and should prove an interesting vehicle for further research into parallel languages.

CNS-1 is an application specific computer, and an important component of the project is the development of software libraries for the applications we have in mind. These libraries should allow much of the peak performance of the machine to be made available to connectionist researchers in a straightforward and flexible manner. These connectionist libraries will be based on connectionist simulators that have evolved over several generations at ICSI, including parallel versions for the RAP.

The capabilities of CNS-1 might go unused unless a comfortable programming environment is developed. In particular, we see the need for powerful debugging tools to detect programming errors and accurate profiling tools to discover performance bottlenecks. The development of CNS-1 software is a major task that must be performed together with hardware design. Accurate hardware simulators will be made available for software development and feedback during hardware development. These should also be useful after hardware design as accurate performance predictors and debugging tools.

Finally, as noted earlier, the use of a standard ISA makes the development of a useful computer by an academic research group much more feasible, due to the large body of available commercial and public domain software that has been written for this ISA.

5 Final Remarks

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 subsystem(s) 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. Experience with the RAP machine on full applications over the last few years has shown us much about these factors, and has guided our design of the CNS-1.

6 Acknowledgments

In addition to the authors, the primary CNS-1 design team members are Tim Callahan, Bertrand Irissou, Brian Kingsbury, Phil Kohn, John Lazzaro, Thomas Schwair, Carlo Séquin, and David Stoutamire. The National Science Foundation provided financial support for through Grant No. MIP-8922354, 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 No. N00014-92-J-1617 and the International Computer Science Institute.

References

[1] 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.

[2] H. Bourlard and N. Morgan. *Connectionist Speech Recognition: a Hybrid Approach.* Kluwer Academic Press, 1993.

[3] Jerome A. Feldman, Chu-Cheow Lim, and Thomas Rauber. The Shared-Memory Langauge pSather on a Distributed-Memory Multiprocessor. In *Second Workshop on Languages, Compilers, and Run-Time Environments for Distributed-Memory Multiprocessors*, Boulder, Colorado, Sept 30 - Oct 2 1992.

[4] H. P. Graf, R. Janow, D. Henderson, and R. Lee. Reconfigurable Neural Net Chip with 32K Connections. In *Advances in Neural Information Processing Systems 3, Proceedings of the 1991 Conference*, pages 1032–1038. Morgan Kaufmann Publishers, 1992.

[5] D. Hammerstrom. A VLSI architecture for High-Performance, Low-Cost, On-Chip Learning. In *Proceedings of the International Joint Conference on Neural Networks*, pages II–537–543, 1990.

[6] J. Hennessy and D. Patterson. *Computer Architecture a Quantitative Approach*. Morgan Kaufmann, San Mateo, 1990.

[7] John Lazzaro, John Wawrzynek, Misha Mahowald, Massimo Sivilotti, and David Gillespie. Silicon Auditory Processor as Computer Peripherals. In *Advances in Neural Information Processings Systems, Proceedings of the 1992 Conference*, December 1992.

[8] Nelson Morgan, James Beck, Phil Kohn, Jeff Bilmes, Eric Allman, and Jochaim Beer. The Ring Array Processor(RAP): A Multiprocessing Peripheral for Connectionist Applications. *Journal of Parallel and Distributed Computing, Special Issue on Neural Networks*, 14:248–259, 1992.

[9] Stephen M. Omohundro. The Sather Language. Technical report, International Computer Science Institute, Berkeley, Ca., 1991.

[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.

[11] Thorsten von Eicken, David E. Culler, Seth Copen Goldstein, and Klaus Erik Schauser. Active messages: a mechanism for integrated communication and computation. In *Proc. Tenth International Symposium on Computer Architecture*, May 1992.