CNS-1 Architecture Specification

A Connectionist Network Supercomputer

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

TR-93-021 April 1, 1993

Krste Asanović, James Beck, Tim Callahan
Jerry Feldman, Bertrand Irissou, Brian Kingsbury
Phil Kohn, John Lazzaro, Nelson Morgan, David Stoutamire
and John Wawrzynek

R4.989

Preface

In the past two decades, the fields of VLSI systems design and massively parallel computation have grown into mature disciplines. Both fields began as research topics in industrial and academic laboratories; today, they form the core technologies for several large corporations. This report proposes a massively parallel computer, the Connectionist Network Supercomputer (CNS-1), which leverages off these fields. By targeting the computer to connectionist networks and related applications, we can focus on custom chip design and efficient software to achieve performance goals which challenge the best that industry has to offer.

Why are we building such a project in an academic environment? Three different issues have motivated us to design and construct the CNS-1.

The first issue involves the difficulty in applying neural network techniques to very large problems. Our research in the area of speech recognition has shown that current neural algorithms are helpful in solving problems with up to one million parameters. Performing this work cost-effectively required the construction of a smaller custom neurocomputer. Will the same algorithms work for one billion parameters? To answer this question requires a system that can train very large networks one thousand times faster than our current machines. Commercial machines with this capability do not exist.

The second issue relates to the field of computer architecture. A strong distinction, apparent in the publications of the field, exists between projects that are actually realized and those which are merely paper designs. As faculty, staff and students at a leading school of computer architecture, we believe it is important to be involved in the design and construction of applied software and hardware systems.

The third issue is the application of VLSI chip technology to system design. Dealing with the complexity of competitive VLSI design is a daunting challenge for an academic research group. However, several factors have led us to include custom VLSI design as a key element in the CNS-1 project. Technically, only custom VLSI lets us exploit the simplicity and parallelism of connectionist computation. From an academic standpoint, we believe university VLSI programs must undertake design projects that meet and challenge the industrial state of the art, and that these projects must be pushed to fabrication, test, and use in prototype systems. This approach teaches students relevant perspectives on VLSI design, and results in significant contributions to the engineering literature.

The goal of the CNS-1 project is to produce a system with supercomputer performance and flexible software for connectionist computation at moderate cost and on a time scale of a few years. This report outlines why we have chosen this task, why we believe we can achieve our goals and the current state of our design.

About the researchers The computing requirements of several neural network research projects at the International Computer Science Institute motivated our initial work in neurocomputing systems. These projects were directed by Dr. Nelson Morgan, Adjunct Professor of EECS and head of the Realization group at ICSI, and by Dr. Jerry Feldman, Professor of EECS and Director of ICSI. The first neurocomputing projects, which resulted in the Ring Array Processor (RAP) machine, were designed and implemented by ICSI staff professionals James Beck, Phil Kohn, and

Jeff Bilmes, in collaboration with Nelson Morgan. As our neurocomputing goals expanded to incorporate custom VLSI implementations, our research team grew to include other members of the UC Berkeley community.

John Wawrzynek, and graduate students Krste Asanović, Brian Kingsbury, and Bertrand Irissou, have joined James Beck in assuming primary responsibility for the hardware design of T0, a prototype of the CNS-1 neurocomputing node. Nelson Morgan is concerned with the suitability of processor architectures to target applications such as speech recognition. This group, along with postdoctoral researcher John Lazzaro and graduate students Tim Callahan, Stelios Perissakis, and Sven Meier, will also develop the hardware for CNS-1. UCB Professor Carlo Séquin has joined the project in an advisory role, bringing his experience from previous large computing projects at UC Berkeley, including the RISC-I and RISC-II.

The software group has also grown to meet the new demands of the project. Jerry Feldman oversees the design of the system and application software architecture of the machine, in collaboration with Phil Kohn and Dave Johnson. Also involved are graduate students David Bailey, Ben Gomes, Srini Narayaran and David Stoutamire. In addition to the full-time members of the project, many other researchers have played significant roles in the development of RAP, T0 and CNS-1. The RAP user community has provided invaluable feedback and suggestions. ICSI visiting researcher Thomas Schwair has taken on an important role in defining test strategies for T0 and CNS-1. Steve Omohundro of ICSI and visiting researcher Silvia Mueller have made key suggestions about the CNS-1 architecture and software.

About this document We have written this document for several reasons. We are now at the point in the project where we have settled the major architectural decisions and we wish to get feedback from other researchers. Unlike an industrial project, we have nothing to lose and everything to gain by sharing our ideas with others. We will also use this document internally as an informal design specification to orient new members of the team. We will continue to add details to it as we proceed. As with many large design projects the *formal* specification will ultimately be embedded in the descriptions written for simulation purposes.

This document is divided into three major parts. Part I provides the technical and motivational background. We briefly introduce the state of the art in neurocomputing and show how our work fits in. Several target applications are discussed and we present a set of performance requirements for our system. Part I closes with an overview of the CNS-1 machine hardware and software. Part II is a technical specification for the CNS-1 hardware architecture represented as a "snapshot" of our current thinking. Since it reflects work in progress, it must not be considered complete. Part III is an initial specification of the CNS-1 software. While this architecture document emphasizes hardware, the software specification is provided as a sanity check and to help readers understand the functionality of CNS-1. The three parts are preceded by an Executive Summary.

Acknowledgments Primary funding for this project since May 1992 has come from the Office of Naval Research, Clifford Lau, program director, URI-N00014-92-J-1672. Major support also comes from ICSI, whose funds are provided by the ministries of research of Germany, Italy, and Switzerland, and cooperating companies. Additional support comes from the National Science Foundation, grants MIP-8922354, MIP-8958568, Graduate Fellowships, and Infrastructure Grant number CDA-8722788.

Executive Summary

The Project The CNS-1 project is an effort to build a real machine to run real programs which solve real problems. Although the project is based at an academic institution, our emphasis is on producing tangible results within a limited time frame and with a realistic budget. Along the way, we expect to raise and answer important questions in the fields of connectionist networks, VLSI design, computer architecture, system software design, and application frameworks for connectionist computation.

Aside from the CNS-1's role as a research vehicle, it is expected to fill a need in the area of connectionist applications. Recent advances in neural net research are outpacing the limits of the fastest available workstations and special purpose neurocomputing systems are becoming increasingly popular. ICSI has been successful at developing such systems over the past 4 years. Neurocomputers under development at other sites are examined in Section 1.4.

Connectionist applications have been written for general purpose supercomputers, but the match is sub-optimal. Current supercomputers are typically engineered to solve problems requiring high numerical precision and wide dynamic range, while many neural network algorithms allow reduced limits on both. We expect to take advantage of this and other application-specific considerations to produce a machine with supercomputer performance but with a major savings in cost and complexity.

The Applications We have explored a number of application targets for the CNS-1, including tasks in speech and language processing, auditory modeling, early and high-level vision, VLSI functional simulation, and knowledge representation. Extensive work in the area of connectionist speech understanding points to the need for a machine two to three orders of magnitude faster than the best machines available today. In condensed form, a statement of the requirements for one abstract problem, representative of potential applications is: *Evaluate the activations in a network with one million units having an average of a thousand connections per unit for a total of a billion connections. This should be done one hundred times per second.*

The Machine The CNS-1 is a multiprocessor system designed for the moderate precision fixed point operations used extensively in connectionist network calculations. Custom VLSI digital processors employ an on-chip vector coprocessor unit tailored for neural network calculations and controlled by a RISC scalar CPU. One processor and associated commercial DRAM comprise a node, which is connected in a mesh topology with other nodes to establish a MIMD array. One edge of the communications mesh is reserved for attaching various I/O devices, which connect via a custom network adaptor chip. The CNS-1 operates as a compute server and one I/O port is used for connecting to a host workstation.

Four of the VLSI processing nodes are connected on a small module circuit board. Sixteen such boards are arranged on the exterior surface of an upright octagonal tower with network connections between boards. The tower provides mechanical support, along with clock, power and cooling resources. One hoop of sixteen boards provides 64 nodes; up to 16 hoops can be stacked on the vertical tower for a total of 1024 nodes. The mesh network connections provide a cylindrical topology, with the bottom edge of the net used for the I/O ports.

The Node Each node consists of one VLSI custom processor and 16 DRAM chips. Using 18 Mb devices, this provides 32 MB total of local memory, plus error correction. The DRAM is connected using four independent Rambus* interfaces. The Rambus channel transfers address, data and control information at 500 MB/s using a synchronous, block-oriented protocol.** No additional logic is necessary to complete a node, improving reliability and minimizing the board area required per processor.

The Processor The CNS-1 custom processor chip, called Torrent, is designed for high performance on connectionist algorithms and other data-parallel codes. The Torrent processor includes a 32b RISC CPU, a pair of vector arithmetic pipelines and a vector memory unit, all able to operate simultaneously. Parallel pipes, along with a 2 GB/s datapath to memory, allow sustained execution of 10^9 multiply-accumulate operations per second at a clock rate of 125 MHz. The RISC scalar unit, based on the industry standard MIPS-I instruction set architecture, is included to provide general computational support as well as address generation and loop control for the vector unit.

To support the large instruction bandwidth, Torrent has an on-chip 4 KB I-cache. A data cache is included on-chip to compensate for the latency in accessing DRAM. To sustain high overall bandwidth, the D-cache is non-blocking, allowing the processors to continue execution from the instruction stream while earlier misses are being serviced. The D-cache also merges multiple scalar accesses into blocks for greater efficiency. An additional feature of the architecture allows the data cache to be bypassed when accessing vector data to avoid cache pollution.

The Memory CNS-1 node memory is implemented using four Rambus channels, each with four 18 Mb RDRAMs for a total of 32 MB. With a *peak* channel rate of 500 MB/s, we anticipate *actual* performance in excess of 1 GB/s aggregate (per Torrent) for block transfers of 32B. Although the RDRAM is based on dynamic RAM technology, the design incorporates several on-chip 1 KB caches to improve the effective access time. Other DRAM-based memory technologies are being investigated (cached and synchronous DRAM) as a hedge against technical or availability problems with the RDRAMs.

The Network Torrent includes an on-chip hardware router to handle message-based communications in the CNS-1 mesh. A fast processor-network interface supports efficient execution of neural networks even when sparse interconnections are mapped across several nodes. To improve efficiency, the message length is relatively short, and low-overhead access directly to the processor registers is provided in Torrent. The active message model is directly supported, where an arriving message triggers execution of a local event handler.

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

---

*Rambus and RDRAM are registered trademarks of Rambus Inc.

**An alternate design using synchronous DRAM is also under investigation.

The I/O Connection between the CNS-1 and external devices is 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 send and receive buffers that can be accessed using a conventional parallel interface. Additional external components can be used to customize an I/O node to provide a wide variety of interfaces, including industry standards such as HIPPI and SCSI. This strategy also allows the construction of custom interfaces for sensors and actuators.

The Diagnostic Network 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. The diagnostic network follows the same mesh topology as the data network, thereby giving redundant paths to any node. The two networks share the same connectors, simplifying wiring.

The Software Since the CNS-1 operates as an attached coprocessor to a host workstation, it will not use a standard operating system nor will it support multiple users. It will, however, include multiple programming and operating environments, to allow systems and applications programmers to work at several levels of abstraction. At every stage of software design, we expect to take advantage of available tools and systems which are compatible with the MIPS-I instruction set architecture. In particular, open systems compilers and debuggers will be extended to implement the Torrent vector unit instructions. Additionally, commercial computers from MIPS-compatible vendors will be used during code development and as simulation accelerators for the VLSI chip design.

Users with mainstream connectionist applications can use CNSim, an object-oriented, graphical high-level interface to the CNS-1 environment. Those with more complicated applications can use one of several high-level programming languages (C, C++, Sather), and access a complete set of hand-coded assembler subroutine libraries for connectionist applications. Simulation, debugging and profiling tools will be available to aid both types of users.

Additional tools are available for the systems programmer to code at a low level for maximum performance. Access to the low-level processor and network functions will be provided, along with the evaluation tools needed to complement the process.

Performance Each Torrent processor, operating at 125 MHz, can perform two billion 32b arithmetic operations per second in the vector unit. This corresponds to one billion connections per second, with a connection consisting of a multiply and an add operation. The memory bandwidth required to support this rate is available from the on-chip instruction and data caches. The off-chip memory access rate of >1 GB/s would typically not limit the raw performance, assuming moderate reuse of operands.

The initial CNS-1 implementation will include 128 processing nodes, giving a maximum computational capacity of 256 billion integer operations per second and a total of 4 GB of storage. The design is scalable to 1024 processing nodes for a total of up to 2×10^{12} operations per second and 32 GB of RAM.

Part I

Introduction

1 Neurocomputing

Connectionist, or "neural" computation is structured as the interconnection of many simple functional units. Over the last 30 years, algorithms based on this style of computing have been used for a variety of difficult applications. For instance, in the 1960's these approaches were used for problems such as lunar terrain analysis and brainwave sleep state classification [Vig70]. More recently, connectionist algorithms have been applied with success to such problems as Hidden Markov model probability estimation for continuous speech recognition [MB90], zip code recognition [MBLD92], and spoken language identification [MC91].

This section discusses directions for future neurocomputing research and the reasons for building custom connectionist systems.

1.1 Motivations

Much of the recent success of the connectionist approach is due to the explosion in computing power available to researchers. This technological advance has been matched by new theoretical work as well as a substantial body of application-oriented research. The synergy between these developments is encouraging experiments with networks and tasks that were not feasible even a few years ago. However, now that scientists and engineers are aware of the potential of connectionist methods, they typically find workstation power to be insufficient by several orders of magnitude. Conventional supercomputers are prohibitively expensive to purchase and maintain for many applications and lack the appropriate software. For these reasons, a number of researchers around the world are engineering supercomputers targeted for connectionist algorithms.

In a university-based research environment, there are only two justifications for undertaking an ambitious development project: either the result is needed, and cannot be obtained elsewhere, or the effort is necessary to advance science and technology. Both motivations are important for CNS-1. Recent work has shown the practicality of connectionist systems for a range of important problems, but has also revealed the need for computational resources far exceeding those available to investigators in the field. The realization of the RAP system at ICSI [MBK+92] is an example of an application-specific project which has yielded results crucial to our research, easily justifying the development effort. In addition, the RAP has proven beneficial to other groups that have acquired the system.

In the case of the CNS-1 project, there are already* interesting results at several levels of hardware and software architecture. These results originate from our examination of the connectionist model of computation, which differs from more conventional models in several important ways. These differences allow us to design and build systems that are much simpler than conventional supercomputers and yet outperform them on the problems of interest to us. In addition, the design choices that work well for connectionist systems appear to be applicable to a broader class of computations and also yield insights into the general problems of parallel hardware and software development.

*Start date for the project was May 1, 1992.

Why are connectionist algorithms of interest? There are a number of answers to this, and their order of relevance is strongly task-dependent. Viewed as probability estimators, connectionist systems are relatively free of distribution assumptions, an advantage when the distribution is unknown. Viewed as function approximators, such systems can implement arbitrary nonlinear mappings. Viewed as computational systems, connectionist networks provide strong parallelism for the applications that can be represented in this way. This parallelism can then be translated to significant performance advantages with parallel hardware. While it would be irresponsible to say that these approaches are *always* better than alternatives, it is clear as of 1993 that connectionist algorithms are a powerful and useful family of techniques for a variety of problems, and engineers and scientists benefit by having them in their "toolkit."

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. Many problems are highly regular and are well suited to parallel and pipelined execution. Connectionist networks are "embarrassingly" parallel and map nicely to distributed memory machines. Communication is usually multicast and normally only values are sent, eliminating the read latency that plagues most distributed computing. There appears to be no need for memory coherence since global variables are not used.

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 decision does little to compromise our application requirements yet eliminates many of the most complex hardware and software difficulties in parallel computing.

For problems such as speech recognition, complete tasks have so many parameters to optimize that training time is the main impediment to progress, often forcing the researcher to make suboptimal decisions. While such research is computationally intensive, the required power could be provided by one of the many large conventional computers being built. Researchers can often do very well with a conventional vector supercomputer, such as offerings from Cray Research. However, the computations per dollar for these machines is quite low in comparison to what can be achieved with an application-specific architecture. One of the reasons for this is that connectionist computation can be done with moderate precision (commonly 8 bits or less for activations and 16 bits for weights). This not only reduces the size of arithmetic units, but more importantly reduces the amount of memory and memory bandwidth requirements significantly.

Probably the most graphic argument for an application-specific system over a general purpose processor is the RAP. Recent experiments at Berkeley show ICSI's RAP machine to be slightly more effective than a single head Cray X-MP for a variety of connectionist computations at less than 1% of the cost. Both of these machines are orders of magnitude less powerful than the CNS-1.

We estimate that when the CNS-1 becomes functional, it will be 20-100 times more efficient in terms of cost, electrical power requirements, weight, and size, than conventional supercomputers. This is true even for microprocessor-base parallel distributed memory supercomputers such as Intel's Paragon series. Table 1 compares some features of two Paragon models and two sizes of CNS-1 systems. For the CNS-1 price, the estimate is based on a RDRAM parts cost of $20 and a cost for all other components associated with a node at $320. The total parts cost is then multiplied by a factor of 4 to suggest a selling price if CNS-1 were produced commercially.

Table 1: CNS-1 vs. Intel Paragon Comparison				
Machine	Power (kW)	Nodes	Peak Performance	Price (est.)
Paragon XP/S-20	19	192	19.2×10^9	$6,700,000
Paragon XP/S-50	86	672	50.4×10^9	$19,000,000
CNS-1 (small)	2	128	250×10^9	$330,000
CNS-1 (largest)	16	1024	2×10^{12}	$2,600,000

Note: Paragon figures are for double-precision floating point while CNS-1 uses 32-bit integer operations.

Not indicated in the chart are other advantages that accrue from designing a machine specifically for connectionist applications. One particular emphasis in the CNS-1 design is a simple method for attaching external sensors and actuators operating under a soft real-time constraint. Retrofitting such an interface to a conventional supercomputer may be prohibitively complex. Another advantage of a custom design is that the realized performance is routinely a higher percentage of the peak value than on general purpose supercomputers. Put another way, the general purpose supercomputer designers include some features in their machines that are expensive yet do not improve performance on connectionist networks.

In the near future, the greatest challenge to a custom architecture machine for connectionist research comes not from the general-purpose supercomputers, but low cost, high performance workstations. Performance in uniprocessor workstations is improving by as much as 60% per year, and multiprocessor systems are starting to be introduced. We believe, nevertheless, that workstation vendors will concentrate their efforts in areas that will not yield optimal improvements for the connectionist community. In particular, the most favorable pricing (in terms of dollar per operation) is frequently the workstation model designed for the highest volume of sales. Such low-end systems usually feature limited expandability, making them incompatible with our large task requirements.

Justifying the effort to construct the CNS-1 rests on the expected performance advantage of application-specific computing. Briefly, the CNS-1 attains this advantage with:

- Application-optimized processor design.
- Parallel processing units on each silicon die.
- Multiple computing nodes within one chassis.
- Simplified memory system architecture.
- A low-latency high-bandwidth communication network between nodes.
- Flexible, high-performance I/O channels.
- New programming paradigms customized for the machine.
- The exclusion of features (hardware and software) least likely to improve performance on a real application.

Even when academic projects result in implementations that are not faster than commercial computers, they often teach us much that can affect the entire field of computing. An example here at Berkeley was the RISC development. While the RISC I and II chips were not faster than commercial microprocessors at that time, they showed the strengths of this style of computing quite convincingly.

1.2 Connectionist Approaches in Signal Understanding

The human nervous system is still the best example that we have of a complete system to perform complex problems in signal understanding. For decades this system has served as a conceptual model for such tasks as speech and image understanding. Sensory input systems such as the cochlea (inner ear) are roughly imitated in speech recognition systems by the microphone and some simple form of spectral analysis. Phonetic classification and likelihood estimation, assumed to occur in a number of brain structures, is typically done in artificially engineered systems using some kind of probability estimator such as a Gaussian mixture model or a back-propagation trained multilayer perceptron. Assembling phonetic information into meaning is probably done in association areas all over the brain, but this process is little understood. The corresponding engineering approach is usually very primitive, for example using a list of allowable wordpairs. Typical engineering systems also do not permit much feedback from these higher level processes into the earlier decisions, while anatomical pathways exist in the nervous system for many such interactions.

While connectionist approaches are not the only way to implement these different pieces of the signal understanding problem, they are plausible, and offer at least a set of techniques that have not been fully explored (particularly for the more syntactic and semantic cases). These approaches generally tend to be less bound by restrictive assumptions than previous techniques, and permit higher-order interactions. Additionally, connectionist architectures provide a consistent expressive framework for all of the pieces of the problem, in contrast to the current situation in which various components are designed by scientists and engineers from different disciplines, often with different perspectives and jargon. Incorporating connectionist models in all aspects of signal understanding should permit feedforward and feedback between different parts of the system in a much more flexible way than is currently practical.

Researchers in this area must use fully programmable digital computers, as many of the design issues are completely unresolved. With some specialization, (particularly moderate fixed-point wordlengths), fast neurocomputers can provide significant advantages over general purpose computers for this research, while being much more flexible than specialized analog systems. The latter can be more efficient for the modeling of specialized sensors and feature extractors, however. The most useful complete systems will probably require both forms of implementation, and effective and flexible interfaces between the two are a major concern in our project.

1.3 Application Targets

It is difficult to design a parallel machine to be equally effective for all applications. Ideally the design team would carefully examine proposed tasks and determine which features were required to optimize performance and cost for the target applications. Unfortunately, for many cases there is a "chicken and egg" problem; that is, the optimal computer cannot be designed until we have experience with the application, but we cannot gain this experience until we have a machine that can run the application efficiently.

A feasible solution to this dilemma is to observe the requirements for smaller tasks that we believe to be similar to our end goals, as well as to analyze those cases which have no good working precedent. A machine is then designed to perform well on both sets of these problems. Iterations on this process should result in better solutions both at the algorithmic and architectural levels.

We have developed a number of application targets for the CNS-1. These range from obvious extensions of tasks we have run on parallel computers at ICSI through bona fide applications that have never been implemented on fast hardware. In addition, we have developed an abstract target application which we believe to be representative of a class of problems which has withstood conventional analysis.

1.3.1 Speech Processing

Over the last few years, we and others have demonstrated that fairly large connectionist networks (more than one million parameters) are effective for the estimation of posterior class probabilities given training data that is a sparse sampling of a highly dimensioned feature space. Specifically, we have been training large multilayer perceptrons to estimate phonetic probabilities for continuous speech recognition. This is of interest because the resulting estimates are provably discriminant (that is, trained to differentiate between different classes) and are also essentially free of distributional assumptions. Recognizers using this mechanism have now been tested on large standard databases, (principally DARPA's Resource Management corpus), and have been found to be at least competitive with more standard approaches while requiring fewer careful optimizations for a particular task or vocabulary [RMCF92].

The Resource Management task incorporates a 109-speaker training set with a total of about 3 hours of speech. This translates to over a million feature vectors of 10 milliseconds each. These features are used to train networks with approximately one million parameters, requiring 5-10 iterations through the training set. The resulting training problem requires on the order of 10^{14} arithmetic operations. Depending on the details of the run, this requires a few days of computing on our current parallel computer, the RAP.

New tasks we are attacking require significantly more computing for a number of reasons:

- The databases we are considering for future research have ten times more training data than the Resource Management corpus.
- We are planning further research in robust front ends, which cannot be evaluated without complete training and recognition iterations. Therefore, a 1-5 day training time is not practical for such an optimization; a one hour turnaround is a more workable goal.
- Network size has increased inexorably over the last few years and the trend will continue. Before the RAP, our nets were typically under 100,000 connections. Our current speech recognition nets range from 300,000 to 1.5 million connections.

When taken together, these increased requirements suggest a computing engine that is 2 to 3 orders of magnitude more powerful than the RAP. This translates to 4 to 5 orders of magnitude faster than a Sparc 2 workstation, or perhaps 3 orders of magnitude more powerful than a 1995-vintage workstation.

In addition to greater raw performance, larger networks and increases in training set size, the CNS-1 includes a more diversified instruction mix to handle the "non-neural" operations. One potentially tricky area is the ability to easily couple dynamic programming steps and the network evaluations without excessive communication overhead. This requires that dynamic programming and pointer bookkeeping must be handled *within* a CNS-1 node.

A final, more speculative point is that as nets grow larger they tend to be more sparsely interconnected. For huge nets, full connectivity is not reasonable, even if the computational power were available, because we lack sufficient data to train the large number of parameters. 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 for the speech recognition research application.

1.3.2 Other Concrete Tasks

In addition to the speech connectionist applications of CNS-1, we are 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. In some cases, such as speech recognition, we already have clear ideas on how to apply 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 intriguing is the prospect that a connectionist supercomputer will encourage explorations on unwieldy problems previously ignored.

We expect many applications will run efficiently on the CNS-1. The following lists some we have considered during the design.

Language processing Speech recognition systems such as the one referred to above have largely ignored language processing above the phonetic level. Once syntactic, semantic and pragmatic knowledge is incorporated, the connectionist computation will expand significantly for the speech understanding task. Connectionist language processing models are under development [Sha88].

Auditory modeling Auditory models such as the correlogram [Lyo92] are instantiations of known physiological or psychological functions. Full implementations require an enormous amount of computation, and researchers would like to couple modifications of these subsystems with speech training systems so that the effect of model changes could be evaluated. This is of particular interest for speech degraded by noise and reverberation.

Early vision An example task is texture recognition [MP89], which requires no learning (the convolution kernels are assumed). For the case of NTSC video and 100 kernels, 100 billion connections per second are required. There is considerable work at UCB being done in this area.

High-level vision This task involves recognition and analysis of objects from some earlier low-level image processing. In addition to the large raw computational requirement, high-level vision is believed to require the sparse interconnection (fan-in and fan-out averaging 100-1000) of hundreds of thousands of units. This is a research focus of the ICSI applications group.

Functional simulation of specialized hardware (VLSI) The design of extremely fast specialized hardware for connectionist computation requires many simulations at the switch or logical level. For this application, the interconnection is typically extremely sparse, since local connectivity is limited in planar silicon architectures. Functional simulation fits well with a major thrust of the EECS department at UCB.

Large conceptual knowledge representation studies Many tasks, including high-level vision and language processing described above, rely on large fully associative memories, like human memory. Simulating such rich conceptual systems was a goal of the original Connection Machine development, and remains a goal of current research.

1.3.3 A Benchmark Problem

In addition to the concrete tasks described above, 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 one million units and an average of one thousand connections per unit for a total of one billion connections. This should be done 100 times per second.*

This description is deceptively simple, and does not specify the distribution of connections. Without the connection context, it is not possible to relate this benchmark to the previously described concrete tasks. Nevertheless, it does allow us to examine the relationship between the connectivity and performance for alternate design choices. In particular, some extreme connection distribution examples must be examined to ensure that the CNS-1 performance is not unacceptably degraded.

1.3.4 CNS-1 Goals

Given the above target tasks and perspectives, we conclude this section with a summary of CNS-1 performance goals:

- Connectionist compute power. The minimum configuration machine should evaluate 100 billion connections per second, which requires a weight-reading bandwidth of 200 GB/s for 2B 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 require a broadcast of 10^6 activations 100 times per second. Each node must then read 10^8 bytes/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 GB are needed to hold one billion 2-byte connections. When activation tables and pointers for sparse networks are included, another 2 GB are required. This 4 GB represents a minimum, and larger CNS-1 systems will be in demand for retaining the largest data sets.
- Sparseness. A network with arbitrary sparseness will run on CNS-1, although with reduced efficiency compared to applications with fully connected subnets. The performance degradation with progressively random sparseness should be gradual, so that a fully random interconnection pattern should be evaluated at a speed that is no worse than 1% of peak.
- 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, even in the case of a large available memory shared weights are a 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. Many applications require integrating operations that are not fixed point vector operations. The CNS-1 node must be able to perform general scalar processing at a rate only moderately reduced from the peak rates of vector operations.
- Floating point support. During the development stages of fixed point algorithms, it is important to be able to write code with multiple precision and floating point representations. Additionally, the final form of an algorithm may include occasional floating point or double precision fixed point variables. The CNS-1 processing node will contain enhancements to allow handling such variables with a reasonable performance trade-off.
- 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 most common functions. However, another design goal is the graceful performance degradation for the general (non-parallel) operations included in the application code. Provision for clean mechanisms of expression for these programming modes is another key requirement for the system.
- Soft real time capabilities. While hard (deterministic) timing guarantees are probably not required for CNS-1 applications, an important aspect of the machine's intended function is the ability to interface interactively with analog sensors and activators. For instance, real-time streams of speech and video images must be accepted by the machine and processed within the appropriate frame times. These time frames are 33 msec for video, 10 msec for preprocessed speech, and 60 µs for raw speech. There should be a simple high-performance interface mechanism that can be used to connect CNS-1 to a variety of target external devices, such as analog circuits that implement complex auditory models.

1.4 Previous Neurocomputers

A number of parallel computers have been built in the past with connectionist computations as the target task. Many of these have used special-purpose architectures incorporating commercial DSP chips. Examples of this approach are the NeuroTurbo from Nagoya University [IYM+89] and the Ring Array Processor (RAP) machine that has been developed at ICSI in Berkeley. In the latter case, the ring-based machine has been used since 1990 as an essential component in the development of connectionist algorithms for speech recognition. Implementations of this machine consist of 4 to 40 Texas Instruments TMS320C30 floating-point DSP chips connected via a ring of Xilinx programmable gate arrays, each implementing a simple two-register data pipeline.

Several related efforts are underway to construct programmable digital neurocomputers, most notably the CNAPS chip from Adaptive Solutions [Ham90] and the MA-16 chip from Siemens [RBR+91]. Adaptive Solutions provides a SIMD array with 64 processing elements per chip, in a system with four chips on a board controlled by a common microcode sequencer. As with the CNS-1, processing elements are similar to general purpose DSPs with reduced precision multipliers. Unlike the CNS-1 hardware, the Adaptive Solutions chip provides on-chip SRAM sufficient to hold 128K 16b weights but has no support for off-chip memory. Larger networks require additional processor chips to obtain the required weight storage.

Like the CNS-1 hardware, the MA-16 leverages the high density and low cost of commercial memory parts. This chip is a direct realization of three general network formulae that summarize many connectionist computations. The system that is envisioned will consist of a 2D systolic array containing 256 of these chips, and the resulting system provides impressive raw peak throughput. Multiple Motorola 68040s with additional integer ALUs are used as general-purpose processors to complement the systolic processing array.

Important goals in the CNS-1 design are to achieve high performance on large, irregular network structures and to tightly couple the scalar non-connectionist portions of real applications with efficient connectionist computation. Both the CNAPS and the MA-16 are designed to be cascaded into larger SIMD processor arrays with only auxiliary scalar capabilities; the CNS-1 is MIMD with integrated scalar units.

1.5 The CNS-1 System Overview

The 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 details of the processing nodes and the communication mechanisms. A custom VLSI digital chip is tailored for neural-network calculations, with integral vector processing units yielding up to two billion integer operations per second. Computing 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 CNS-1 implementation will include 128 processing nodes, giving a maximum computational capacity of 256 billion integer operations per second and a total of 4 GB of storage. The design is scalable to 1024 processing nodes for a total of up to 2×10^{12} operations per second and 32 GB of RAM. One edge of the communications mesh is reserved for attaching I/O devices, allowing up to 8 GB/s of I/O bandwidth. The CNS-1 system is attached as a compute server to a host workstation.

Each processing node comprises a custom single chip processor, called Torrent, and DRAM chips. No external logic is necessary to complete a node, increasing reliability and minimizing the board area required per processor. Four complete Torrent nodes are placed on a circuit board module, which is attached to the outside surface of an octagonal cylinder (Figure 1). Clock, power and cooling resources come from the center of the cylinder, while the communication network connections are made along the cylinder surface. By placing the modules on the cylinder surface, network cabling is kept simple and communication distances are minimized.

The Torrent chip includes a 32b RISC scalar unit compatible with the industry standard MIPS-I instruction set architecture. This RISC CPU provides tightly coupled scalar processing for those applications which have a significant non-neural component. The main compute engine in Torrent is a vector unit with eight parallel 32b datapaths, where each datapath can complete two arithmetic operations per cycle. To provide the high memory bandwidth needed to keep the vector unit busy, the memory interface implements four Rambus channels [Ram92] with an aggregate memory bandwidth of well over 1 GB/s. The Torrent nodes in CNS-1 have four 18Mb RDRAM*

*__R__ambus __D__ynamic __R__andom __A__ccess __M__emory.

chips per Rambus channel giving a per node total of 32 MB of RDRAM.* 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. On-chip instruction cache helps provide the required instruction bandwidth.

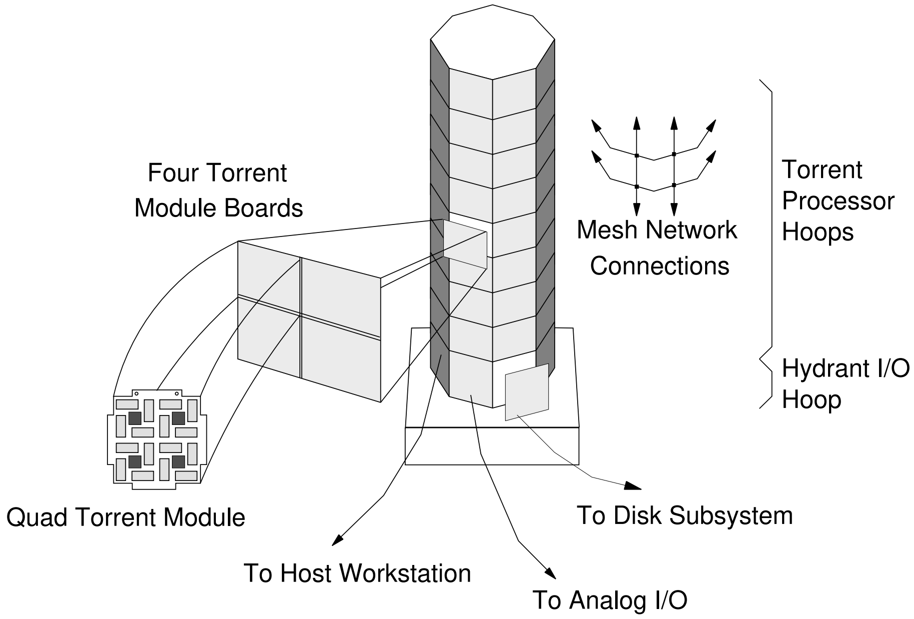

Figure 1: CNS-1 Hardware Overview

Torrent includes an on-chip hardware router to handle communications in the CNS-1 network. An important feature of Torrent is a fast processor-network interface tailored for effective execution of connectionist networks with sparse interconnections and activations. To handle communication of continuous external asynchronous sensory data streams, Torrent hardware implements a short message protocol with access directly to processor registers. However, larger block transfers can be programmed in software with minimal CPU burden due to low-overhead event handling. This event handling is based on the active message model [vEC+92], with an arriving message triggering execution of a local event handler routine. This mechanism is similar in spirit to message-driven architectures [DW89], but requires simpler hardware and decouples message handling from computation.

A host computer and other devices 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. For custom interfaces to sensors and actuators, field

*Alternative memory system designs using wider busses with conventional DRAMs are also being considered.

programmable gate arrays may be used as the peripheral interfaces to Hydrant. 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 requiring 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 acknowledgement wire in each direction. The data network is synchronous with a clock rate of 125 MHz, giving a peak data throughput of 125 MB/s in each direction per link. The largest CNS-1 system with 1024 Torrent nodes has a bisection bandwidth of 8 GB/s, or 8 MB/s per processor.

A separate bit-serial network (TSIP) 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. The network wiring is implemented using the same connectors and cables as the data network mesh.

Much of the software for CNS-1 will be developed with the help of industry standard compilers and debuggers for MIPS-I compatible machines. At the lowest level of software development will be diagnostic routines for detecting hardware failures in the machine. These routines are written for the host system and use the diagnostic network to control the CNS-1 for the duration of the diagnostic tests.

During normal operation, users start their application from the host by invoking the CNS server program (`cnsserver`) and giving it the name of a CNS-1 executable. Assuming the machine is free, the server process first resets each node and then downloads and starts an executable program. This program will usually be the same for each node (SPMD) but may be different on different nodes (MIMD). Next, `cnsserver` enters a monitoring loop, repeatedly scanning for error flags and profiling information across the diagnostic network. In the early CNS-1 system, the host is also used for operating system support, 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 high-level languages will be supported by CNS-1. The commercially available MIPS languages (C, C++, FORTRAN, Pascal, Ada, etc.) will be supported along with an object oriented language developed at ICSI, Sather [Omo91]. In addition, a Torrent assembler filter will be written to help in the development of the optimized libraries.

Support for parallel languages on distributed memory machines is still very much a research issue. Current efforts to port our parallel version of Sather, called pSather [FLR92], to the CM-5 should provide valuable lessons in this area. 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 system, and an important component of the project is the development of software libraries for the applications we have in mind. These libraries should allow a high percentage 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 simulators that have evolved over several generations at ICSI, including parallel versions for the RAP.

Part II

Hardware

2 Hardware

This section presents the hardware design of the CNS-1. The two custom VLSI circuits, the Torrent processor and the Hydrant network adaptor are described first. Next follows a description of the two internal networks used in the CNS-1, the data and diagnostic nets. Lastly, the physical design issues are introduced, with emphasis on clocking, power distribution, packaging and cooling details.

2.1 Torrent Processor Node

The Torrent processor node provides the computational power of the CNS-1. Each node includes the Torrent processor, DRAM chips and communication channels for the interconnection of several nodes.

2.1.1 Torrent Chip Overview

Torrent is a single chip processor designed for high performance on connectionist algorithms and other data parallel codes. The processor is a highly pipelined vector architecture based on the MIPS-I instruction set architecture (ISA) and designed to run at 125 MHz. Torrent integrates a 32b integer scalar unit, two fixed point vector execution pipelines, a vector memory pipeline, on-chip instruction and data caches, a 1 GB/s off-chip memory interface, and a network interface with hardware routing which supports up to 1 GB/s of data I/O. The Torrent chip is designed to form part of a large scale multiprocessor system and requires minimal support circuitry to implement a processor node. Torrent also includes a serial diagnostic network interface used for bootstrapping and debugging.

The Torrent processor for the CNS-1 is an *instance* of a more general design, described in the Torrent Architecture Manual [Asa93a]. Another instance of the same architecture is **T0**, to be fabricated before Torrent [Asa93b]. T0 is designed without a network interface and will use conventional SRAM storage on a compute accelerator board for a workstation.

2.1.2 Instruction Execution

Torrent initiates the execution of a 32b instruction every 8 ns (for a 125 MHz clock rate). Each instruction adheres to the MIPS-I ISA specification, with CNS-1 extensions added within the coprocessor framework. The instructions are dispatched to one of four execution pipelines: the scalar unit, vector memory pipelines (VM), or one of the two vector arithmetic pipelines (VP0 and VP1).

The Torrent ISA extensions define approximately 100 new instructions in five categories:

- Vector Control Register. Used for establishing configuration and monitoring vector unit operation.
- Vector Branch. Modifies CPU control flow based on vector comparison.
- Vector Load/Store and Move. Allow moving vectors of data between vector registers and memory or between sets of vector registers.
- Vector Integer Arithmetic. Include arithmetic, shift, logical and conditional operations on vector register contents.

- Vector Fixed-Point Arithmetic. Provide scaled and rounded fixed-point arithmetic operations on vector register contents.

Details of the instruction dispatch and execution are under development. The general model, however, is that Torrent is superpipelined in each of the execution units, with branch and load delay slots where appropriate. The vector units can begin a new instruction every 4 cycles, and when appropriately combined with scalar instructions, sustain the peak performance. Data accesses are non-blocking and can bypass the cache to allow high off-chip memory bandwidths to be maintained.

2.1.3 Scalar Unit

The scalar unit provides general purpose computation and overall control of Torrent, as well as support for the vector unit. Based on the MIPS-I ISA, the scalar unit is binary compatible with computers using MIPS R2000 and R3000 processors. The interface to the vector unit is implemented as a MIPS-standard coprocessor CP2. To simplify synchronizing the scalar unit and vector memory pipelines, Torrent includes the SYNC instruction from the MIPS-II specification.

The scalar unit includes a fully bypassed register file with 31 32b general purpose registers plus **`r0`**, which is hardwired to the constant 0. In addition, three special purpose registers are included for MIPS compatibility: the program counter (**`pc`**) and two registers to hold the results of integer multiplies and divides, **`lo`** and **`hi`**. These special registers are used or modified implicitly by certain instructions.

The CPU functional units include a 32b adder, a 32b logical unit, a 32b shifter, a 32b×32b multiplier-divider, a zero comparator, the program counter datapath and an address adder. Also included in the scalar unit is the standard system control coprocessor CP0 used for memory management and exception handling.

2.1.4 Vector Unit

The vector unit consists of eight identical datapaths with two execution pipelines in each datapath. (Figure 2) The execution pipelines, VP0 and VP1 are similar, and share access to a common register file. Each pipeline contains a 32b logical unit, a 32b adder, a 32b shifter, a zero comparator and a clip unit. Additionally, VP0 contains a 16b×16b multiplier, pipelined to produce one result per clock cycle. VP1 also includes a population count unit, and a CNTLS (count leading signs) unit.

The vector register file in the vector unit, also split among the 8 datapaths, holds 16 or 32 vectors. (Figure 3) Each vector contains at least 32 elements, of 32b each, up to a maximum of 512 elements.* Vector register **`v0`** is defined to hold the constant zero. Note that a minimum register file design, with 16 vectors of 32 elements each, requires 2 KB of storage.

Twenty-one non-datapath registers have been defined in the vector unit, divided into two categories. Five registers are defined for specific control and status functions and are accessed with the CFC2 and CTC2 instructions. The remaining 16 registers in the vector unit are general purpose, and can be loaded and stored from either the CPU register file or directly from memory. These general purpose registers are used primarily for two classes of operations:

---

*The final vector register file architecture will depend on algorithm requirements and silicon area constraints.

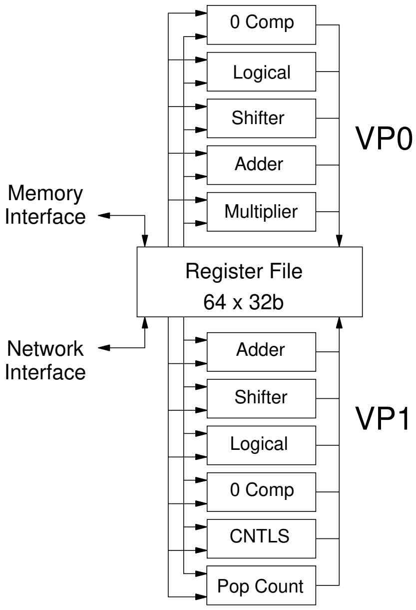

Figure 2: One Vector Datapath

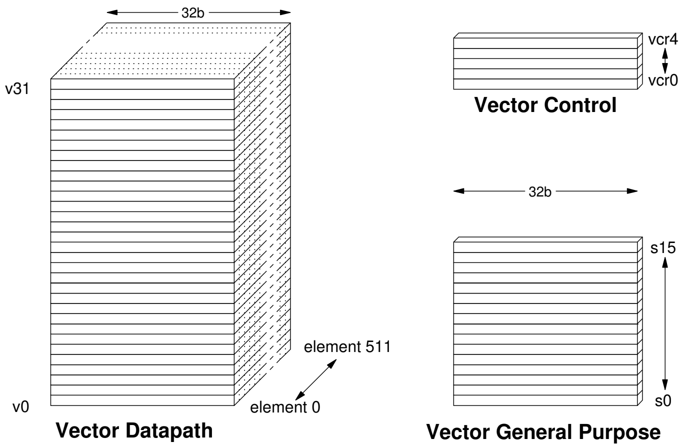

Figure 3: Vector Unit Registers

- Vector datapath configuration. These allow control of the vector datapath's shift, clip and round functions for the most common multiply-add operations. By providing this level of hardware flexibility, the maximum precision in the integer vector units is obtained without sacrificing performance.
- Scalar operand registers. For vector-scalar operations, registers supply the scalar values.

2.1.5 Vector Memory Pipeline

The vector memory pipeline can move vectors between registers and memory as well as perform moves from one vector register to another. Vector loads and stores transfer words, halfwords and bytes (32b, 16b and 8b) between vector register locations and memory. Bytes and halfwords are sign-extended when loaded into the vector elements. Memory address generation for vector loads and stores can use one of three techniques: unit-stride, arbitrary stride, and vector indexed.

The vector move instruction transfers elements between pairs of vector registers. One of the scalar registers in the vector unit provides the index into the source register, and the instruction provides the address of the destination vector register.

2.1.6 Floating Point Arithmetic Support

Although floating point arithmetic on CNS-1 is implemented in software, the vector unit in Torrent contains a small amount of hardware to support floating point calculations. Each datapath contains a "count leading signs" (CNTLS) unit that takes two inputs, X and Y, and returns a count of the number of bits in X (scanning from most significant bit) equal to the sign of Y. This unit accelerates the normalization step required by floating point addition and multiplication routines.

The shifters in the vector datapaths are also customized for floating point support. Each of the shifters can use separate shift amount values, allowing the binary point alignment step of floating point normalization to occur in parallel. A more conventional design, with a single shift amount value, would not provide the parallel performance speedup.

These two modifications are inexpensive, requiring minimal design effort and increasing the size of the vector unit by about 5%. However, by including them in the design, floating point performance is significantly improved. Without this support the vector unit performs 32 floating point adds in 1680 clock cycles; with this support it performs 32 floating point adds in 264 clock cycles.

2.1.7 Rambus Interface

Torrent applications operating on large data sets will require high off-chip memory bandwidth. To supply the bandwidth necessary to feed the vector unit, Torrent parallels 4 Rambus interfaces. The arrangement of the Rambus ports in Torrent is shown in Figure 4.

The Rambus protocol transfers data in blocks, with larger blocks giving greater effective bandwidth. To simplify the protocol, the Torrent Rambus interfaces (RBIs) only transfer naturally aligned blocks of 8B, 16B, or 32B of data. Using a 32B block size gives a data bandwidth on RDRAM cache read hits of over 250 MB/s, with a latency of approximately 110 ns.

The large number of RDRAMs in the CNS-1 system represents a considerable reliability concern. Each RDRAM includes a ninth data bit paralleling every byte, allowing us to implement error detection and correction (EDC). Single bit error correction and double bit error detection will be performed on the 8B blocks over a single port, and will be managed by the port RBI. Correctable errors will be fixed and written back to memory, on the assumption they are transient. Hard (uncorrectable) errors are considered catastrophic and cause Torrent to reset into bootstrap mode.

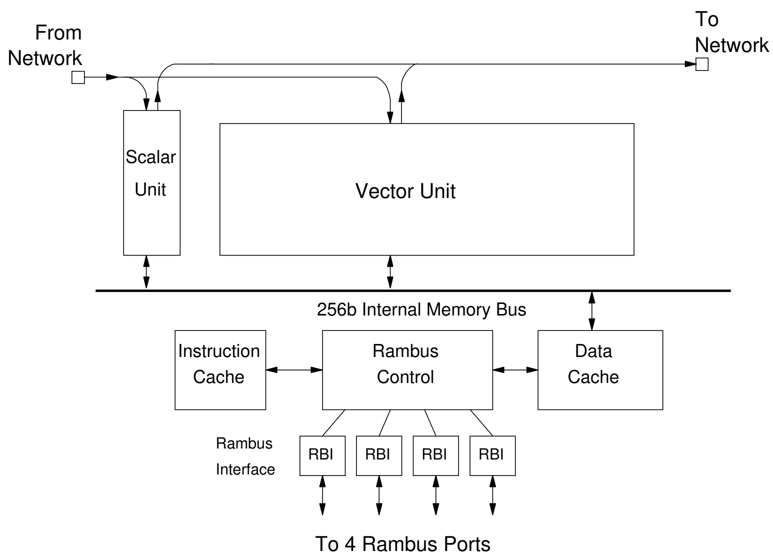

Figure 4: Torrent Chip Block Diagram

Each RBI has two sticky register bits per RDRAM bank to indicate if any correctable or non-correctable errors have been found in that bank. These sticky bits can be read over the serial interface port as well as by the processor in bootstrap mode, allowing Rambus and RDRAM faults to be located easily by diagnostic software.

An alternative design is also under investigation what would use synchronous DRAM rather than RDRAM. This is a more conventional approach allowing a simpler interface, running at more conservative data rates.* To attain the performance goals, wider off-chip data busses are required.

2.1.8 Instruction Cache (I-cache)

To support the large instruction bandwidth required, Torrent has an on-chip I-cache of 4 KB, or 1024 instructions. The I-cache supports one 32b instruction fetch per cycle (500 MB/s). The I-cache is direct mapped with 32 lines, each holding a 20b tag, a valid bit, and 32 instructions (128B).

On an I-cache miss, the instruction fetch phase of the CPU pipe is stalled while the I-cache is refilled. An instruction read request is sent to the Rambus controller, which, after allowing ongoing Rambus transactions to complete, eventually returns the 128B I-cache line. At best, an I-cache miss takes 20 cycles (160 ns) to service: 1 cycle to transmit the instruction read request, 14 cycles for the fastest Rambus read when the port is free, 4 cycles to return instruction responses from the Rambus controller, and 1 cycle to finish writing the last block to the I-cache. If the Rambus is busy when the I-cache miss occurs, the service time may increase by an additional 11 cycles.

*Synchronous DRAM runs at 10 ns/B versus 2 ns/B for RDRAM.

2.1.9 Data Cache (D-cache)

To help overcome the relatively long DRAM latency (14 clocks for a Rambus read), Torrent includes a fast data cache on chip. The D-cache supports up to one 256b data access per cycle (4 GB/s), while also effectively merging multiple scalar accesses into block accesses. To support applications with poor cache behavior, every Torrent memory operation has an "allocate" flag that indicates if the data should be allocated in the primary cache on a cache miss. This prevents data which is unlikely to be reused from polluting the cache.

Two features help Torrent sustain high memory bandwidth. The first is a decoupled memory system including a non-blocking D-cache. This allows the CPU to continue execution while data is fetched from off-chip memory, so multiple memory requests can be issued. The second is a set of vector load/store operations that transfer aligned 32B blocks of memory treated as 32×8b, 16×16b or 8×32b operands. Although the cache bus can transmit a complete 256b block every cycle, there is only a single memory port on each register file, limiting vector load/stores to transferring 8 operands per cycle. Further instructions can execute while a multicycle vector load/store operations takes place, provided they do not include another memory operation.

2.1.10 Torrent Events

The on-chip network interface and timer use the Torrent asynchronous event model. Torrent is designed to handle these asynchronous events quickly, and typically only a few overhead cycles are required from the CPU to service an event. The overhead cycles are used to write a return instruction pointer to a fixed register in coprocessor CP0, and may also transfer further data into both the scalar and vector register files.

The three kinds of events on Torrent are message arrival (inlet), message departure (outlet), and timeout (timelet). The message arrival event is set *pending* when a new message arrives on the node, and becomes *active* when the inlet handler is started. The outlet handler is similarly invoked when a network output buffer becomes available. The hardware timer generates the timelet event when the counter reaches the value specified in the timeout register.

The active message model [vEC+92] allows event handlers to be executed within the current compute thread without requiring pipeline flushes. By keeping event handlers short and scheduling related computation for background (i.e. non-event) processing, the network can be serviced with acceptable latency without excessive context swapping. Event handlers will execute within reserved registers, allowing compute threads to run until they yield, minimizing register spills and cache misses. The Torrent system coprocessor CP0 also includes control bits to disable event handling during critical sections, allowing synchronization between compute threads and event handlers.

2.1.11 Network Interface

The network interface supports a 2-dimensional mesh topology, with 4 bidirectional links to neighboring processors. Each link consists of 2 bytewide unidirectional connections, providing up to 125 MB/s in each direction for an aggregate node I/O bandwidth of 1 GB/s. Deadlock-free, cut-through routing is supported in hardware, with message buffering to allow high link utilization.

The processor-network interface is designed to be general, simple, and fast. Torrent sends and receives messages in registers, with a small amount of network buffering provided at each port. A send instruction specifies up to 5 32b scalar registers as well as a 32b instruction pointer, together with an optional data block obtained from a vector register. The message is then moved from the message buffers into the network under the control of the network router. Similarly, an arriving message is first moved from the network unit into datapath message buffers. The message can then be transferred directly from the buffers into the register file, stealing a few cycles from the CPU to complete the operation.

Active messages are supported in hardware, allowing fast, custom handlers to be directly invoked by arriving messages. If active messages are enabled, the processor will generate an event whose instruction pointer is the head of the message. By software convention, the message arrival frame pointer is generally the next word in the message.

2.1.12 Timer

Torrent has an on-chip timer consisting of a 32b counter cycling at the processor clock frequency. A system timeout register together with a comparator allows a timeout event to be generated when the counter reaches the preset value. The timeout instruction pointer is specified in a system register and a software convention names a scalar unit register to hold the frame pointer. This hardware can be used to implement various timer facilities in software. Since the timers on all Torrents are reset and clocked synchronously, a valid global timestamp is maintained.

2.1.13 Torrent Silicon Technology

Torrent and Hydrant are both full custom integrated circuits, fabricated through the MOSIS* service. It is anticipated the CNS-1 chips will use the MOSIS 0.8μm scaled CMOS design rules, although the 0.5μm process will also be considered. The design uses a combination of full-custom cells for the datapaths and pads, and standard cells for control and other random logic. Supply voltage for the chips will be 3.3 volts, making them compatible with the RDRAM supply.

The silicon tool set used for the CNS-1 project includes a mixture of commercial and public domain software. We are using `magic` for layout editing, HSPICE and CAzM for circuit level simulation, `irsim` for switch level simulation, ViewLogic for schematic capture, and the Lager system for logic synthesis.

The silicon design style used for the CNS-1 chips has evolved over a three year period, with several test devices fabricated along the way. One crucial design choice is the use of the "True Single Phase Clocking" (TSPC) method for latches and registers [YS89]. TSPC designs result in reduced complexity, higher density, and higher speeds than conventional CMOS clocking methodologies. The CNS-1 chips will each use a single central clock buffer to help reduce clock skews associated with conventional clock buffer chains. An on-chip PLL (Phase-Locked Loop) will be used to reduce skews between chips by compensating for temperature and voltage variations.

*MOSIS is the broker for the chip; fabrication will be done by Hewlett-Packard.

2.2 Hydrant I/O Chip

Hydrant is a network adaptor chip that allows external hardware to be attached to the CNS-1 network. Hydrant has the same network interface hardware as the Torrent processor with an internal router controlling 4 bidirectional links, each supporting 125 MB/s in each direction. In addition, Hydrant has a TTL/CMOS level interface called the H-port. The H-port includes control plus a 64b bidirectional data bus that can cycle at up to 50 MHz (400 MB/s).

Hydrant has two message send buffers and two message receive buffers. As a message is being sent from one buffer to the network, another message can be written into the other buffer. The H-port has chip select, read/write, address, and transfer size control lines to provide a simple memory mapped interface to the message buffers. The H-port data bus supports 8b, 16b, and 32b accesses as well as full 64b accesses to simplify attaching slower speed peripherals. Message flow control is managed on the H-port with four asynchronous handshake lines, one in each direction for both send and receive. Hydrant also includes a Torrent serial interface port (TSIP) connection for diagnostics, system reset, configuration, and monitoring. The H-port signals can be controlled and sampled over TSIP.

Simple peripheral interconnects may be handled with a small amount of external glue logic (for example a Xilinx FPGA) connected to a commercial transceiver chip. More complicated peripheral protocols (such as SCSI) may require a dedicated commercial processor on the I/O card. The H-port can then be memory mapped into this I/O processor's address space. In the same manner, Hydrant can be attached to any commercial processor to allow these to be added as processing nodes in a CNS-1 system.

Hydrant will also be used to attach experimental sensors and actuators directly to CNS-1. A custom sensor/actuator chip may integrate an H-port I/O interface on-chip. The different H-port data transfer widths allow the designer a tradeoff between high bandwidth and low pin count.

2.3 Data Network

The CNS-1 data network is a point-to-point, nearest neighbor, packet-based grid interconnect linking Torrent processing nodes. This section describes the data network and presents the results of simulation studies showing network behavior.

2.3.1 Topology

Each Torrent node includes network connections to its four nearest neighbors, forming a mesh. This mesh is circularly connected in one dimension to form a cylindrical topology. Torrent nodes at the top and bottom rim of the cylinder are connected to only three neighbors, with the fourth network connection self-terminated. The cylindrical mesh topology was chosen for several reasons:

- A mesh is a direct network of constant degree, and can be implemented within the Torrent processor chip. No additional router devices are required, significantly reducing the complexity and physical size of the machine.
- The wiring between nodes follows a simple physical mapping with uniformly short connections. Short wires allow high data rates with low power consumption.

- The mesh is simple to scale across a machine of the targeted size. Within the range of 128-1024 nodes, the cylindrical mesh requires a maximum of 47 network hops. For a random distribution of communication packets, the average number of hops for the largest machine is under 20. For connectionist algorithms, network locality will be exploited to significantly reduce the number of hops.
- The mesh gives very high bandwidths for those algorithms that map well, including low-level image processing and dense matrix manipulations such as those found in many neural network algorithms. The bisection bandwidth per processor ranges from 16 MB/s in the smaller systems to 8 MB/s in the largest CNS-1 system.

The decision to use a cylinder topology rather than connecting the ends to form a torus is based on the difficulty of implementing the electrical interconnect. Constructing a physical torus in three dimensions is unrealistic, and remapping a torus to a simpler shape is also problematic. A common approach for such a mapping requires cables to make connections to “next-to-nearest” neighbors, bypassing the nearest neighbor nodes completely.

Although the number of cables for the remapped torus and cylinder designs are approximately the same, the *length* of each cable is increased by more than a factor of two for the torus. This extra cable slows signal propagation between nodes while increasing the size of the module boards. In addition, having to snake cables around neighbor boards is a mechanical nightmare, and interferes with cooling, power distribution and the clocking arrangement. By adopting a cylindrical form for the *mechanical* design of the CNS-1, implementing the cylindrical *network* interconnect is very economical.

2.3.2 Physical Link Interface

Neighboring nodes in the mesh are connected by a bidirectional link. The wires in the link are unidirectional, with 8b of data and an acknowledgement wire in each direction. The network is synchronous and runs at the processor clock frequency of 125 MHz, giving 125 MB/s of peak data throughput in each direction. The acknowledge wires are used for hardware flow control.

The network signalling uses a low-voltage swing, and is source terminated to match the network line impedance. The network signals are carried between processor modules on flexible circuits that include ground planes to reduce crosstalk and maintain a controlled impedance path.

2.3.3 Message Format

CNS-1 messages consist of two header words, followed by 0-512 words of data (0-2048B), and a trailing CRC check byte. The header contains the destination address and the length of the data section. The network address is split into 2 dimensions corresponding to the row and column location of the destination on the mesh. The header also contains a short tag that is not interpreted by the routing hardware. The tag is used by Torrent to distinguish the different formats of messages it can send and receive, including the length of the scalar portion of a message.

The check byte holds a CRC signature calculated over the whole packet and is used to detect transmission errors. Each router keeps sticky error bits for each input port, and these can be monitored over the diagnostic network. The network should be reliable under normal operation and parity errors are considered catastrophic. If Torrent receives a corrupted message it will jump back into bootstrap mode.

2.3.4 Routing

The CNS-1 uses a fixed (non-adaptive) routing protocol with absolute node addresses. Packets from a given source to a given destination are always routed over the same links, regardless of the traffic encountered. The routing protocol is simply stated as:

- First route from the source node around the ring (horizontally) to the correct column, using the shortest path. In cases where the destination column is exactly opposite the source node, the direction chosen (clockwise or counterclockwise) alternates with the column address of the source node.
- Next route vertically within the column to the correct destination node (Figure 5).

2.3.5 Buffering and Deadlock Prevention

If the topology were a simple mesh with no wrap-around in either dimension, it would be simple to avoid deadlock: always route in one dimension first, and then in the other dimension. However, the connection of the nodes into rings in the horizontal dimension makes deadlock possible unless proper precautions are taken, since there are now cycles present in the channel dependency graph [DS86].

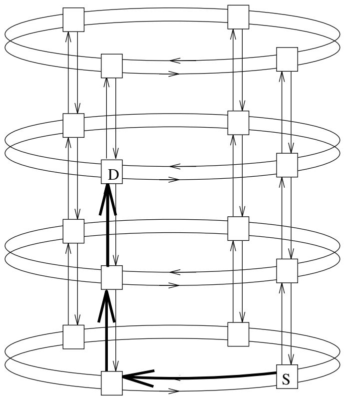

Figure 5: Network Routing Protocol

The method recommended by Dally and Seitz in this situation is to map two or more virtual channels onto each physical channel, and then connect the virtual channels in such a way that no cycle is formed. However, we have chosen a simpler method which is just as effective in our situation. This method utilizes output buffering, with a dedicated FIFO for each output link (Figure 6). When a packet is blocked, it will be stored in the output FIFO, thereby staying out of the way of messages coming in from behind which are destined for other output links.

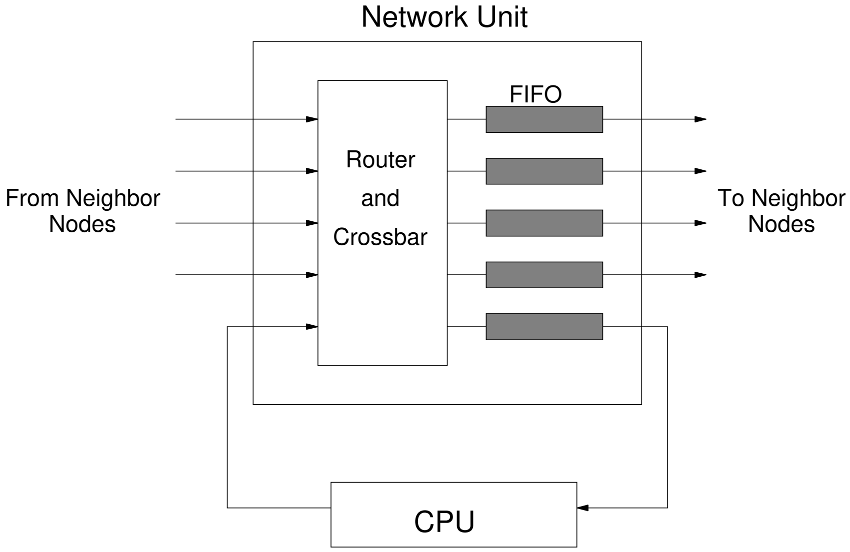

Figure 6: Data Network Interface

The output buffers are “cut-through” FIFOs, which means that a packet will start leaving the FIFO one clock cycle after entering if the output link is available. This greatly reduces latency compared to “store-and-forward” buffers.

The deadlock prevention technique used in CNS-1 is closely related to the buffering scheme. Briefly, deadlock prevention is based on the processors guaranteeing that they will eventually accept packets even when they cannot send them. With the fixed routing protocol used in the CNS-1, once a packet is on the second phase of its journey (traveling up or down a column), it cannot contribute to a deadlock. This is because packets traveling on a column never try to route back onto a ring, so such packets will never be part of a cycle of deadlocked packets.

The result is that deadlocks can only occur around a ring. Such a deadlock would occur when the buffers in one direction around the ring are all completely full — no data can leave one of the buffers for the next node, because the next node has no space for it. However, as long as there is empty space in at least *one* of the FIFOs around the ring, data can still move, (albeit slowly) eventually clearing the possible deadlock.

This conclusion leads to the anti-deadlock requirement: at least one buffer in a ring must be “not full.” Data words already on the ring and continuing their journey will not cause a problem, since for every such word that comes into a node, a new space has opened up in the sending node. Only new packets from processors can potentially cause deadlock. When launching a new packet, it must be guaranteed that even after the packet is completely in the ring, there is still at least some empty space in *some* buffer around that ring. The easiest way to ensure this is to make sure that there will be space left in the *local* buffer. This leads to the following restriction:

Before a network interface routes a packet from the local processor to a horizontal output, it must wait until there is ***more*** *than enough room in that output FIFO for the entire packet.*

2.3.6 Message Priority

The routing hardware is responsible for selecting which of several waiting input packet to route on a given link. Although some restrictions are placed on this decision to avoid deadlock, there are many cases where two or more inputs are eligible to be routed to an output. Priorities assigned to the different input links are used to resolve such cases.

For a horizontal (around ring) output link, both continuing traffic already on the ring and new packets from the processor can compete. Continuing packets are always given priority, even if there is enough room in the output buffer for the packet from the processor.

For a vertical (up or down) output link, three types of packets compete: continuing vertical packets, turning packets which are coming off a horizontal ring, and new packets from the processor. Packets turning off of a ring always get highest priority, packets continuing vertically next highest, and new packets from the processor get lowest priority.

This priority strategy is justified by the fact that for most traffic distributions and all but the largest CNS-1 systems, the horizontal links (around the rings) are the most utilized. Since they are the most likely communication bottleneck, it is critical that the horizontal rings are not stalled due to blocked packets. It is more acceptable if packets traveling vertically encounter slight delays, since on average there is less traffic in that direction.

2.3.7 Performance

Although the concept of virtual channels was originally useful in preventing deadlock, Dally also showed that the use of virtual channels increases the utilization of the physical links, thereby increasing network throughput [Dal90]. Simulation of CNS-1 systems from 128 to 1024 nodes and with various traffic distributions have shown that the simple output buffering scheme described here has throughput and latency characteristics at least as good as the virtual channel/wormhole routing scheme described by Dally. This is likely due to the fact that our packets are small; virtual channel/wormhole routing is more appropriate for larger packets. As an additional advantage, our scheme will be simpler to implement.

2.4 TSIP Diagnostic Network

The CNS-1 diagnostic network is a configurable serial path running through every node in the machine. The network is under control of the host processor, and performs several functions related to testing, system initialization, and performance monitoring. A major design decision in the project was to *not* use the industry JTAG standard for the diagnostic network, but to design an enhanced version, the Torrent Serial Interface Port (TSIP). In the following, the required functions of the diagnostic network are examined, along with reasons JTAG was judged unsuitable for CNS-1.

2.4.1 Diagnostic Strategy

The largest version of CNS-1 will require over 17,000 active devices* to be manufactured and tested on over 256 circuit boards. At an estimated 850 solder joints and 100 mechanical connections per

*Over 2 m^2 of silicon will be needed, containing more than 300 *billion* transistors.

node, the full system has nearly one million places where *something can go wrong*. Exacerbating the situation is the difficulty in repairing faulty modules. The Torrent node will use removable connectors for the networks, power supplies and central clocking, but all components on the module board will be attached with solder.

Faced with these realities, the strategy for manufacturing reliable module boards is crucial to the success of the CNS-1 system. Once reliable, debugged module boards are installed in the system, the diagnostic emphasis turns to operating system and applications development. Taken together, these requirements suggest the following diagnostic strategy:

- The Torrent processor chip will undergo a multiple step test procedure, starting at the wafer level and ending only after the burned-in module board is installed in a working system. These tests are primarily based on a serial scan technique.
- Commercial RDRAMs will have extremely low failure rates. Although RDRAMs do not include boundary scan capabilities, conventional memory testing should pinpoint faulty parts or erroneous assembly.
- Circuit connections between boards will be tested directly by neighboring Torrent processors executing test routines.
- The Torrent chip will include several checkpoint levels of operation after the power is applied during the boot procedure. One crucial capability will be to allow Torrent to treat the caches as memory and operate directly, without attached RDRAMs.

2.4.2 Diagnostic Port

On the Torrent chip, TSIP consists of four serial interface ports with an internal configurable router and a hardware protocol handler. The wiring consists of a pair of signals in each of the four directions paralleling the fast data network connections. Each pair includes one data-in and one data-out signal.

In normal operation, only two of the four ports are used, configured as a single serial scan path through the chip. By providing ports on all four sides of Torrent, TSIP supports multiple redundant paths to any node. All communication on the TSIP network is local between adjacent Torrent or Hydrant nodes, and the only global signal required is the system clock.

TSIP sends information as serial packets along a chain of connected nodes under the control of the host processor. These packets contain either route connection commands or data and control for conventional diagnostic functions. The network routing is established at power-up as part of a diagnostic probe procedure, and remains static unless there is a link fault. Each TSIP packet contains a 4-bit control token and is optionally followed by 32b of data during instruction register or data register shift operations. The control tokens include commands to configure routing, capture local instruction and data values into shift registers, shift the instruction and data shift registers, and update local values from the shift registers.

Making TSIP packet-based eliminates the JTAG requirement for a global function control signal, but also makes it more difficult to synchronously capture data across a complete system. Since TSIP passes packets down a serial data link, the last node in the chain will receive a token after the first. To compensate for this and allow synchronous data capture, TSIP includes a decrementing counter in the interface on each node. As part of the configuration process, the host presets the counter with a value which allows for the packet transit delay. When a node sees a "capture" token it begins to count down from this preset value, and will capture data on the cycle when the counter reaches zero.

2.4.3 TSIP-JTAG Comparison

The industry standard JTAG boundary scan specification (IEEE 1149.1) provides solutions to many test problems by incorporating serial scanning circuits on every pin of a device. At a minimum, JTAG allows checking circuit board connections between devices, as well as providing limited access to internal chip functionality. JTAG may be optionally extended to include sophisticated built-in self-test (BIST) or other diagnostic operations. The JTAG *framework* is certainly flexible enough to encompass the range of operations required for CNS-1.

The implementation of JTAG, however, is more of a problem for CNS-1. The standard requires two global signals, posing a serious engineering challenge over a large machine such as the CNS-1. Separating the signals into "regional global" areas is possible, but requires resynchronizing circuits at the region boundaries. Global signals are also more prone to single point failures, a disadvantage in a diagnostic network.

Another difficulty with the standard (for CNS-1) is the function of boundary scan. According to the JTAG standard, all device pins should be controllable from the serial scan path, allowing test vectors to be transferred between chips. This "controllability" at the device boundary allows testing circuit board traces between JTAG compliant devices at low speed. In CNS-1, over half of the active Torrent chip connections are made to RDRAMs, which are not JTAG compliant. All of the remaining Torrent signals, with the exception of CLOCK, are connected to neighboring Torrents.

As a result of these obstacles, the CNS-1 diagnostic network will not be JTAG-compliant. By incorporating the functionality of JTAG in TSIP, however, we expect to leverage off previous work done during the development of the standard.

2.4.4 Additional TSIP Operations

Additional utility functions will be provided by the host over TSIP. These functions include system reset, configuration, and bootstrapping. A single bit data register per Torrent is included to allow selective resetting of individual nodes. All Torrents and Hydrants will undergo a configuration phase using TSIP to establish network size and node addresses.

To run an application, the system must be reset and then bootstrapped into full operation. When Torrent is reset, it enters level 0 bootstrap mode with the caches locked to perform as on-chip memory. The instruction cache is then loaded with a bootstrap program over TSIP, and Torrent released to execute the bootstrap.

When the system is running, errors may be detected by hardware. Examples include memory parity errors, message parity errors, and out of range message addresses. These errors may indicate hardware faults and should terminate the current program run. Flag bits are included on a TSIP scan path to allow the host to monitor possible error conditions by repeatedly scanning. The host can then translate the location of the error on the scan chain to the position of the faulty part in the system.

Hardware performance monitoring (HPM) is a technique that has been successfully applied to help tune application codes for supercomputers, as well as to gather statistics to help future architecture designs. Some of the advantages of HPM over software profiling are its speed, non-intrusive nature, and accuracy. We expect to include HPM features to allow sampling Torrent state using the TSIP. From this data the host can build a statistical picture of the machine's behavior.

2.5 Physical Design and Implementation

Except for the two custom silicon devices, the design of CNS-1 relies on commercially available components. The following sections detail current plans for the physical design of CNS-1.

2.5.1 Packaging Overview

The CNS-1 physical design is a blend of traditional and unconventional approaches to computer design. One unconventional aspect of the CNS-1 is the macroscopic package form. The traditional computer form of cabinets with backplanes holding large circuit boards, cooled by several local fans, has been abandoned for a radically different form. The regularity of a message-passing architecture gives us the ability to choose a different machine form; our requirements for high node density and economic implementation push us to exploit this ability.

The CNS-1 is envisioned as an upright octagon-shaped tower. The tower is formed by stacking octagonal "hoops" of processors; each hoop consists of 16 identical circuit modules, each containing 4 Torrent processors and memory. The circuit modules are connected to each other with flexible cables. This module arrangement naturally implements the cylindrical mesh connection topology of CNS-1.

The hoops stack onto an octagonal frame that provides physical support, as well as a conduit for wiring and cool air. The column is mounted in place of a computer-room floor tile to allow forced air to enter the bottom of the tower. The top of the tower is sealed forcing air to exit the side of the tower through cooling holes in each circuit module in each hoop.

Up to 16 hoops fit onto a single tower, for a system with 1024 nodes. The minimum system will consist of a single hoop, providing 64 nodes. The bottom edge of the lowest hoop is reserved for connecting custom I/O boards. These boards connect to the mesh network on the upper side, and a special channel interface cable on the lower side.

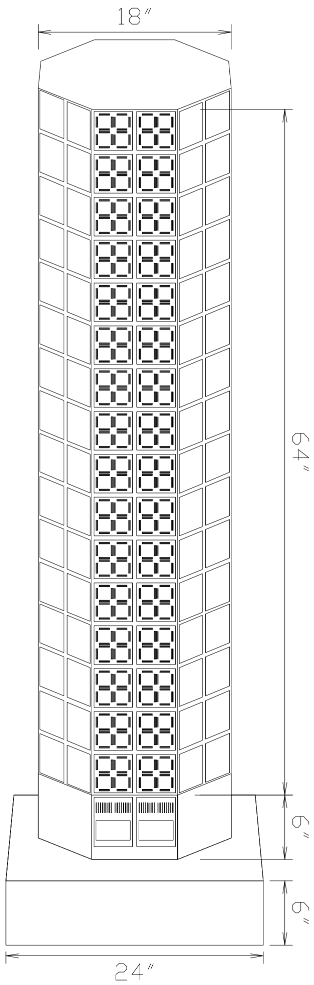

Figure 7: Torrent Tower.

2.5.2 Module and Wiring Technologies

Torrent circuit modules consist of 4 Torrent processors and the associated memory, connectors for flexible wiring to neighboring modules, power and clock connectors, passive components for resistive pullups and power supply bypassing, and heat sinks. The module technology will dictate the size of the module, and indirectly the height and diameter of a Torrent tower. Other characteristics of CNS-1 affected by the module technology are reliability and heat dissipation.

Two module implementation technologies are currently under consideration: multilayer printed circuit boards (PCB), and laminate-based multichip modules (MCM-L). The tighter tolerances of the MCM-L technology enable a more aggressive (higher performance and smaller size) module design than with conventional PCBs. Figure 8 shows a preliminary Torrent module layout using MCM-L, shown actual size.

The inter-module wiring technology is also under review at this time. Several types of flexible wiring and connector technologies are being evaluated with respect to electrical properties, reliability, and cost. Power and clock connectors are also under review.

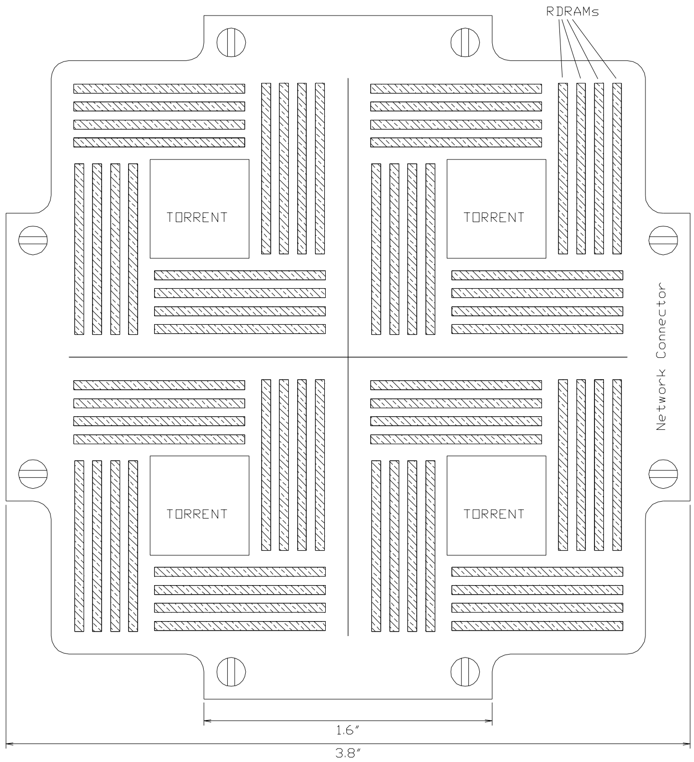

Figure 8: Quad Torrent Module

2.5.3 Clock Distribution

The CNS-1 will implement a simple brute force global clocking method for completely synchronous machine operation. A single clock source (variable during debugging, replaced with a crystal in the functional machine) will propagate through a buffer tree optimized to match the physical layout. This global clock strategy greatly simplifies the network link control, and phase control logic (phase locked loops or variable phase delays) can be eliminated entirely. We feel this plan will be successful for the following reasons:

- The cycle time of the communication link (8 ns) is relatively long for the VLSI technology used. This "slow" cycle translates to more tolerance of the skew from the clock buffer tree.
- Full advantage will be taken of the physical arrangement and symmetry of CNS-1. Physical arrangement in the sense that the clocks propagate from one end of the machine to the opposite end with no loopback. Clock skews along this axis are of concern only for communications between adjacent processing nodes, and will be minimal. The radial symmetry of the cylinder allows simple balancing of loads and distances to further reduce skew.

The device used in the clock buffer tree will be similar to currently available GaAs PLL/clock drivers from Vitesse or Triquint. Both manufacturers have devices which provide controlled skew between the input and multiple outputs, along with frequency multiplication and small time adjustment controls. It is anticipated that even better devices will be available by the time the CNS-1 design is frozen.

Physically, the buffer chips in the clock spine will reside along the central axis of the CNS-1. There they will be located in the cooling airstream, improving the delay matching between the parts. Radial connections from this central spine will be made through the octagon sidewall to the individual module boards using coaxial cables.

2.5.4 Heat Removal

The CNS-1 is expected to operate from a standard 120 VAC, 20 Amp circuit in the smaller configurations only. With a single processing node requiring between 10 and 15 watts, a 128 node system would need up to 2 kW, about the power of large copy machine. The heat produced will require continuous air cooling for proper operation. Larger systems would operate from a 230 VAC line, with the largest (1024 node) system requiring approximately 16 kW of power.

For the 1024 node system, chilled air is drawn in through the bottom of the base cabinet, either with the help of an internal fan, or simply channeled from the underfloor plenum of the computer room. After passing through a filter, the air is introduced into the bottom of the central octagon. The upper end of the octagon is closed. Beneath each processor chip location, a circular exit hole drilled in the wall of the octagon allows cool air to escape the central plenum.

The processor module board, containing four Torrent chips, includes four heat sinks mounted just "under" (toward the center of the octagon) each of the chips. These heat sinks are situated so that when the module boards are attached to the exterior face of the octagon, the air exiting the octagon impinges the heat sink directly. By using the central octagon as a pressurized plenum, each processor is assured of a supply of cool air without regard for mounting location.

Once the cool air has passed over the processor heat sinks, it moves away from the center, through gaps between the boards. This "waste" air is contained by an outer cylinder (also octagonal) and constrained to flow up through an opening at the top. This airflow will contribute primarily to the cooling of the RDRAMs, which will be situated directly in this airstream.

Analysis A common calculation for airflow requirements is:

$$V = 1.76 \times P/\Delta T$$

where: V is volume of airflow in cu ft/min; P is power, in Watts, of device to be cooled; ΔT is temperature rise allowed, in degrees C. The constant in the equation is for sea level air and must be increased for higher elevations. A common safety factor is a 25% margin over the calculated value to allow for decreases in fan or filter efficiency over time.

With the simplifying assumption that this airflow will directly only cool the Torrent chips (at approximately 10W each), and not the RDRAMs, we obtain an airflow of 1 cu ft/min, or approximately 29 cu in/sec. (The waste air will cool the RDRAM, requiring a separate temperature rise calculation.) With an exit hole area of 1.5 sq in, this yields a linear flow of approximately 20 in/sec through the wall of the octagon. The total volume requirement for the largest system would then be 1024 cu ft/min.

Common "muffin" fans used to cool desktop computers can typically move air at a rate of 50-150 cu ft/min, with low back pressures. Performance of these devices falls off quickly as the pressure increases within the chamber receiving the air. For the largest CNS-1, we will instead use a single reverse impeller fan, with a capacity of approximately 2,000 cu ft/min.

By requiring the large CNS-1 configurations to be situated in a cooled computer room, the power control is automatically linked to the room ambient temperature. This is an important safety feature in the event of a failure of the air conditioning. Nevertheless, the CNS-1 will also include a minimum of one airflow sensor and one temperature sensor. Detection of an out-of-range condition will shut down the processor power, while maintaining the fan operation.

Part III

Software

3 Software

It has traditionally been easier to build special purpose processors than to program them effectively. A great deal of design effort has gone into making CNS-1 programmable by experts and also useable by scientists with minimal interest in low-level coding. This is feasible because much of connectionist computation is expressible in terms of high-level constructs that can be efficiently mapped to parallel hardware. Our group has had considerable success with general connectionist simulators and with specialized simulators oriented towards efficient parallel execution.

3.1 CNS-1 Software Overview

The software sections in this document are more tentative than the hardware because of the different timetables involved. Although a great deal of cross-platform development can be accomplished before the Torrent node is functioning, much coding work will remain after the VLSI task is complete. The primary justification for examining the software picture at this juncture is to ensure that the hardware includes sufficient resources to attain the system goals.

We divide the CNS-1 software description into several sections. Listed from low level to high level these are: diagnostics, `cnsserver,` run-time resource management, programming languages, local and distributed libraries, CNS-1 simulation and debugging, and some typical application primitives.

3.2 Machine Diagnostics

Access to the CNS-1 to perform low-level machine diagnostics is through the TSIP (Section 2.4) network, controlled from the host. A suite of diagnostic software will incrementally check Torrent processors, RDRAM and the network connections using TSIP to pinpoint the location of any hardware faults. Additional diagnostic programs will be written for Torrent that test functionality of the processor and attached Rambus memory. These will be used to verify Torrent die and modules after fabrication, and to test Torrents *in situ* if hardware failures are suspected.

3.3 cnsserver

A Unix workstation acts as host to CNS-1 and runs a `cnsserver` process for the duration of each application run. The `cnsserver` process is responsible for first resetting and bootstrapping CNS-1, downloading the application, and then servicing I/O requests while monitoring errors, finally shutting down the machine before exiting. In addition, the `cnsserver` can be used to gather hardware performance monitoring information.

At the start of every application run, `cnsserver` uses the TSIP network to reset and bootstrap CNS-1. The system reset sequence configures network addresses and clears all hardware error flags. System bootstrapping occurs through a sequence of levels. After reset, each Torrent node is in bootstrap mode, where the on-chip caches are locked and treated as on-chip memory, and the Rambus channels and the network are accessed as peripheral devices. The system boostrap sequence first downloads a small level 0 bootstrap code into each Torrent's I-cache using TSIP. The Torrent node then executes the level 0 bootstrap, whose only function is to download the level 1 bootstrap over the much faster data network. Each Torrent then executes the level 1 bootstrap which initializes the Rambus RDRAMs, and downloads the system kernel and application code over the data network. The final function of the level 1 boostrap is to jump to user mode and to start an idle loop (level 2). The application is started by a message from `cnsserver` to one of the processing nodes. The TSIP implementation is fast enough that this complete bootstrap procedure should only take a few seconds at the start of each application.

One of the scan registers on the TSIP network provides access to CNS-1 state at run-time to allow error monitoring and non-intrusive real-time profiling. After the `cnsserver` has started the application, it continually scans this register while waiting for I/O requests. Memory and message parity error flags are scanned to allow faults to be detected quickly. Profiling information is available in the form of program counters for the Torrent nodes, network and Rambus buffer status bits, and a software writeable register on each Torrent node. By repeatedly sampling this profiling information, the host can collect statistics of processor, memory, and network usage, useful for debugging and performance tuning. Experienced users may derive new versions of `cnsserver` to perform application-specific profiling using the software loadable register.

Programming errors may cause an application to deadlock, possibly filling and blocking the network. In these cases, the user can instruct `cnsserver` to enter debug mode. The TSIP network is used to force all nodes to jump into a debug monitor that is linked in with all application code and protected by the write barrier. This debug monitor disables active messages and reads out all messages in transit into local scratch memory. This will then free the data network to allow debugging commands to be sent to individual nodes.

3.4 Run-time Resource Management

CNS-1 is a single user machine, hence there is little need for conventional operating system software. To support memory protection, Torrent uses a single control register to divide the memory space for user mode access. Addresses below the boundary specified by the protection register may only be accessed in kernel mode; user mode accesses cause an exception.

A set of user-level run-time libraries and software conventions support management of node CPU, memory, and network resources. A few scalar unit registers are reserved for common run-time tasks. Node CPU resources are consumed by threads and event handlers, node memory resources are consumed by activation frames and dynamically allocated data, and network resources are consumed by messages.

3.4.1 CPU Management

The Torrent CPU divides its time between compute threads and event handlers. Compute threads perform the bulk of the work in an application, while event handlers implement global communication, I/O, timers and synchronization between compute threads. At any time a Torrent will be running some compute thread (or an idle thread) which will be interrupted from time to time to run an event handler for network or timer events. Typically, the event handlers are short and use only a few pre-allocated registers, thereby causing minimal disturbance to the ongoing computation.

The context for compute threads is saved in call frames allocated from the heap. The Torrent hardware can support a variety of scheduling disciplines; we describe a scheme based on the TAM model [CSS+91]. In the general scheme, a call frame may have several associated threads, and a thread may use several call frames, depending on the compilation strategy of the supported language.

For example, a simple C compiler would allocate a single thread for each C program and this C thread would use multiple frames, one per procedure invocation. At any time the C thread would have (up to) one active frame. The C thread may get descheduled as a side effect of calling a communication or synchronization library routine.

Threads and frames are the entities which are scheduled onto the CPU, and this scheduling is handled by software conventions built into the compilers and run-time libraries. Frames are the entities which may consume CPU registers when made resident on the CPU, and threads are the entities which may consume CPU time. Compute threads run non-preemptively with respect to each other, and include code to schedule their successor, i.e. as each compute thread completes, it jumps to the next compute thread to be run. If a thread is the last thread attached to the resident frame, it is responsible for saving any live CPU registers before exiting. Similarly, the first executed thread of a newly resident frame is responsible for loading any saved CPU registers.

A frame with any enabled (ready-to-run) threads is placed in an enabled-frame queue. Frames with no enabled threads live in the heap until some event causes them to become enabled and moved into the enabled queue. The basic case has one thread per frame and looks like an ordinary task queue. There is no fixed scheduling policy for placing newly enabled frames into the active queue. A compute thread can choose its successor in any desired manner, and an event handler can place a newly enabled frame at any position in the queue. A simple policy is to allow frames to be placed at the beginning or end of the queue and to always select the head of the queue. For example, a simple C port would always execute the newly enabled frame corresponding to a new procedure call immediately without even consulting the active frame pool, to allow arguments to be passed directly in registers.

Event handlers are short threads that run in response to events occurring on the CPU. The three main classes of events on Torrent are message arrival (inlet), message departure (outlet), and timeout (timelet). The message arrival event occurs when a new message is received from the network. When active messages are enabled, the processor will jump to the event handler specified by the instruction pointer carried in the first message word. The message arrival handler is responsible for integrating the message contents with the ongoing computation on the node. Typically, each message will also include a frame pointer, and the inlet will save some data from the message and update some synchronization counter in the frame, possibly causing a compute thread to be enabled in the frame. If the frame was not already enabled, the frame may then move onto the enabled queue.

The message departure event occurs when Torrent has space in its output buffers for an outgoing message. The address of the outgoing message event handler is stored in a system register on the CPU. The frame pointer for the outlet is usually held in a reserved scalar unit register. The user libraries include asynchronous DMA-style send operations that cause an enabled DMA frame to be appended to the outlet frame queue. At any time, at most a single outlet frame is resident in the CPU. As each outlet event occurs, the handler is called to send out another small piece of data for the currently resident DMA output frame. When the current DMA output is completed, the thread brings in the next DMA output frame from the outlet frame queue. These DMA message sends can be freely intermingled with shorter message sends.

The timeout event indicates that the timer in the CPU has reached the value specified in the CPU timeout register. A CPU system register holds the address of the handler for timer events. The enabled-timer frame queue is kept in time order, with earliest frames first. Each timelet performs the desired timed action, perhaps reinserting itself into the timer queue, then installs the earliest next timer-frame in the timeout register, and the timelet instruction and frame pointer registers. User library routines provide a timer facility, as well as allowing tasks to be scheduled at future times.

3.4.2 Memory Management

The writeable memory on each Torrent node is managed as a single heap. All dynamically allocated data, including procedure call frames, are allocated from this heap.

The most general procedure call frame management strategy is to allocate individual frames from the heap. This allows multiple parallel frame invocations to share the heap area as needed, but may incur a significant run-time overhead even with optimized frame allocation routines. The other extreme of call frame management is to allocate each thread a fixed size stack at thread create time, with compiled-in checks to ensure the stack size is not exceeded. An intermediary solution may allocate stack frames in chunks at a time. This policy need only be fixed on a language by language basis provided debuggers are adapted accordingly.

3.4.3 Network management

The user code is given direct access to the network with no operating system overhead. Prototype code will be available to assist users in developing routines that do not deadlock the network. In addition, the DMA output routines described in Section 3.4.1 will provide efficient large block transfers. In practice, much of the communication usage pattern anticipated for CNS-1 involves multicast and we are developing direct support for this.

3.5 Application Software

One goal of the CNS-1 system is it should be useful to a wide range of people, including those who have no background in computer science and parallel computation. Without the appropriate software support, programming the CNS-1 would be a very difficult, primarily because of the complexity inherent in parallel computing and vector assembly coding. To provide a range of tradeoffs between efficient and flexible use of the hardware, we identify three levels of user within our target community:

- Researcher - the *primary user* of the CNS-1; writes little or no code. Gains access through pre-written applications.
- Developer - codes in C or C++ or Sather using high-level libraries of distributed data structure classes.
- Wizard - uses assembler, C or C++ to control hardware and write libraries.

These three levels provide progressively more complex models of the machine by exposing more details of the hardware and machine configuration. Moving to the higher levels of abstraction provides more specialized functionality along with greater protection from subtle programming errors.

3.5.1 Researcher Model

A **researcher** might employ the CNSim (Section 3.13.3) simulator to build new high-level objects (such as groups of units) by combining features and interconnections with other objects. A graphical user interface will make construction by interconnecting these building blocks more interactive and intuitive. The researcher need never know the CNS-1 is a parallel machine; all parallelism and data distribution is handled by the class libraries used by the applications programs.

3.5.2 Developer Model

A **developer** writes object-oriented code (in C, C++ or Sather) by utilizing high-level libraries of distributed data structure classes (see Section 3.9). Most of the computational loops and interprocessor communications that are required inside these high-level libraries are calls to the low-level libraries written by wizards. Every distributed data object is created on a group of processors, called a *cluster set,* specified by the developer. The developer can view the CNS-1 as a group of these cluster sets. Each cluster set represents some number of processors that can be used together in a single program multiple data (SPMD) manner.

The high-level libraries hide the parallelism within a cluster set by taking care of distributing the data and computation among the processors. A cluster set is not restricted to being used as a single SPMD machine as described above. For example, sometimes it might be useful to have one processor of the group running different code that manages the other processors where global control or calculation is required. Again, these details are hidden in the high-level library.

The developer selects an allocation algorithm from the library that assigns processors to each group. Sometimes it is useful to subdivide the machine in different ways for different phases of a computation. Since processor groups can overlap, during the course of a computation the developer can activate different sets of groups.

3.5.3 Wizard Model

A **wizard** knows *everything* about the machine. Wizards write code for the Torrent nodes in assembler, C and C++. All hardware level code is written by wizards, including the primitives for reset, download and core dump along with system support functions such as block transfer, multicast messages, inter-node synchronization, and I/O. Much of this work involves writing active message handlers. Wizards will also write the computational loops that form the basis of the applications library, such as those outlined in Section 3.13. Torrent programming should not be more difficult than for similar vector processors, and we expect doctoral students to be able to perform CNS-1 wizardry.

3.6 Programming Languages

All compiler, assembler, and linker tools will be developed to run on Unix workstations, using a conventional strategy for compilation and linking. Separately compiled object files are linked into a single executable which is broadcast by `cnsserver` into all Torrent nodes at the start of each application run.

3.6.1 Assembler and C/C++ Compiler

Since Torrent is compatible with the MIPS-I ISA, the existing MIPS compilers and assemblers can be used to generate Torrent code. The added Torrent vector coprocessor CP2 instructions are generated by C++ inlines or C macros that use the "asm" function. The same procedures are also used by the library routines, also written in MIPS assembler language. Since the MIPS assembler would not understand these new opcodes, all assembler code is passed through a filter program that changes them into generic coprocessor CP2 opcodes which *are* supported. This filter program can be easily implemented using a macro processor (such as m4) or a stream text editor (such as sed).

Future projects may include a port of the `gcc` compiler that has a built-in primitive data-type that supports direct manipulation of Torrent vector registers. This would require the user to break up vectors into the correct length for the hardware. There may be some cases where it is worth the trouble to code at this low level in C or C++, but it is not worth the investment of time to write a new assembler language function. Eventually, a fully automatic vectorizing compiler could be developed. Adapting an existing vectorizing compiler may not be too difficult since the vector coprocessor architecture is fairly standard.

3.6.2 Gang-C/C++

A possible next step in the `gcc` port may be to add explicit support for the "gang" data type. A gang is a short fixed size vector used by the Torrent vector register file and function units. This would allow C/C++ code to directly use the vector unit. For the most part, the C/C++ compiler would treat gangs as a special primitive data type (like float or int). In particular, we can prototype this easily using C++ inline assembly methods, and then consider moving to compiler support for increased efficiency.

3.6.3 Sather

Sather should be relatively easy to port to Torrent, since it is translated into C. Adding gang support to Sather might even be easier than C++ because the compiler development is being done at ICSI.

3.6.4 Parallel Languages

Lastly, we could port a true parallel language (such as pSather or a parallel variant of C/C++). This would involve considerable effort and research. A port of pSather to the CM-5 is underway, and should provide useful experience for the CNS-1 pSather port.

3.7 Local Object Libraries

A number of optimized libraries will be developed that operate only on data local to a node. The core routines in these libraries will be coded in assembler.

3.7.1 IEEE Floating Point Support

Torrent will have library support for full IEEE floating point. This support will include scalar floating point code that can be directly used by compilers, and vector floating point operations that will be made available as library routines.

3.7.2 Block Floating Point Support

The optimized libraries to perform fixed point arithmetic are designed to work on a block floating point representation where one exponent is kept for a complete vector or matrix. When these library routines are packaged in suitable class abstractions for C++ or Sather, they should remove much of the difficulty of dealing with fixed point representations. These will be the main libraries used to implement neural net algorithms for the machine.

3.7.3 Timer Libraries

The timer libraries will provide an interface to the on-chip timer facility. Current time may be read to give a globally valid timestamp, and threads may deschedule until a given future time.

3.7.4 Profiling

Profiling will be especially important for CNS-1, since the relationship between the program and the execution time will be much less obvious than for most serial machines. In addition to the hardware profiling scheme outlined in Section 2.4.4, we intend to support more detailed, but intrusive, software profiling.

Software profiling is enabled by compiling code with a special option and linking with profiling versions of the standard libraries. Software profiling keeps track of messages sent and received from each processor, while also implementing standard program counter sampling techniques using the hardware timer. The message logs can be sorted to examined which network links may be limiting overall performance.

3.8 Communication and Control libraries

Libraries for inter-thread communications and synchronization will be provided. Communication libraries will include routines for asynchronous bulk data transfers, broadcast, and multicast. Synchronization libraries will include barrier objects and semaphores. If pSather is ported to CNS-1, the more elaborate monitor primatives will also be available.

3.9 Distributed Object Libraries

The application programmer can manipulate data structures that are spread among a number of processors by using classes defined in the distributed object library (DOL). The root class of DOL manages an array of user objects that is split up among a set of processors. These user objects may be as simple as fixed point numbers in the case of distributed matrices and vectors. More complex distributed data structures, such as sparse matrices or B-trees, involve more complex user objects.

Many functions in the DOL class insulate the user from any knowledge of parallelism. The exact number of processors used by the function is determined at runtime, allowing the same object code to run on different size machines. The DOL class writer is responsible for defining default resource allocation and management policies that will create this illusion for the user. In the cases where performance is most important, the user or a higher-level application can provide hints to the DOL classes such as:

- The percentage of the available processors to use.
- Upper and lower limits on the number of processors required for the function.

- Other DOL objects that should be located nearby.

When complete control is required, the user can specify the explicit mapping of data structure arrays (called "chunks") to processors.

pSather incorporates direct language support for distributed objects. Special data and control structures in the language will allow user code to be run on a set of processors defined by a distributed object. The development of distributed data structures and algorithms is a major focus of the pSather effort and much of this will be exploited in CNS-1.

3.10 Input/Output Libraries

I/O devices are physically connected to the CNS-1 through the I/O ring at the bottom of the machine. The I/O ring consists of pairs of Torrent and Hydrant chips. These I/O Torrents are used to perform I/O related processing, reducing the load on processors in the main array. The Hydrant physically connects high speed devices to the array, translating external events to active messages and vice versa. These device interfaces may include HIPPI, SCSI, SBus, VME, HDTV and other, analog-based interfaces. One of the Hydrant ports is used to connect the host to the CNS-1 data network. In addition, the host has connections to control the TSIP network.

Initially, CNS-1 relies on the host operating system for access to files and host devices, and `cnsserver` will service CNS-1 system calls. For applications requiring very high bandwidth file I/O, a commercial disk array will be directly connected to a Hydrant interface, eliminating the host from the I/O data path. The host will continue to exercise *control* over I/O, however, providing protection services and file management through the Hydrant.

The Hydrants in the I/O ring are also used to attach sensors and actuators for real world I/O, but these devices will generally require a device specific driver. The drivers will use inlets, outlets, and timelets to efficiently couple external events with the ongoing computation. In effect, external hardware is simply treated as another asynchronous parallel task and, in particular, it will be possible to prototype real-world hardware and software by replacing the I/O device with a simulation.

3.11 Simulation Environments

There will be several simulation environments, each focused on a different group of users. At the lowest level, the Torrent chip design process requires a progression of simulations. Although most are too detailed to be of any general use, the instruction set (ISA) simulator will be used for system design and low-level debugging. This simulator efficiently models the behavior of a Torrent chip using a workstation. The ISA simulator is based on SPIM, which claims an average 25 times slow down over native code. A graphical user interface (under X) provides easy access to all internal state and execution control.

The ISA simulator will be adapted to generate program traces and statistics that are useful in optimizing the system design for typical application programs. Design and optimization of the network topology and routing strategies will involve running many ISA simulators on the same workstation. This full CNS-1 simulation may also be useful for analyzing the performance of the network when running actual application programs. Even for very small training runs, this simulation may require using the CM-5.

CNS-1 applications that use the distributed object library can be recompiled to run on any workstation using the simulated DOL. Much of this is the same code as the actual CNS-1 DOL; only the low-level routines that use Torrent vector and network coprocessors are simulated in C.

3.12 Debugging

The initial bootstrap and diagnostic software will be tested using the instruction level simulator for the Torrent ISA. This will be written and tested before the first silicon is available.

Local object library routines only run on a single Torrent processor, and will be debugged using `gdb` ported to the Torrent architecture and on T0. Remote debugging will be used, allowing most of the debugger functionality to be implemented on the host. The node will only run a simple monitor program during debugging. Communication between the two halves of the debugger will be via TSIP, which will include specialized debugging functions.

For low-level distributed software development, multiple `gdb` sessions can be run on the host, targeting individual Torrent nodes. Because TSIP is used as the debugging network, there is no contention with application code for the data network. Communication debugging is done using special message send and receive handlers that keep track of the source, destination and time of each message. This history can be displayed chronologically for the whole machine to reconstruct the sequence of events before a fault.

We do not have a general solution for the problem of debugging high-level parallel programs. There may be specific solutions that are customized for library classes, with debugging features integrated into the objects. The general case, however, remains an area for future research.

3.13 Applications Programming

Since CNS-1 is an application-directed system, much of its design has been based on detailed analysis of the most important anticipated algorithms. Some examples of this kind of study are presented in this section. Our most common algorithm, back-propagation learning in fully connected and fully activated networks, is presented first.

3.13.1 Dense Back-propagation

The computationally expensive ($O(n^2)$) operations of back-propagation can be reduced to the following matrix operations (similar useful primitives can be found for other network formalisms):

- multiplication of a matrix by a vector
- multiplication of the transpose of a matrix by a vector
- addition of the outer-product of two vectors to a matrix

Here we will examine how multiplication of the transpose of a matrix by a vector is achieved using parallelism in various forms on CNS-1. This is perhaps the most important operation, the essential computational mechanism of the forward pass of a neural network.

Matrices are striped across the rows, assigning multiple columns to reside on each processor. Matrix transpose multiplication with a vector copied on all processors therefore becomes multiplication in place on each processor's submatrix. The matrix multiplication is accomplished by each processor multiplying its portion of the complete matrix and doing a reduce-add operation using the network. For the operations required of dense back-propagation, a ring network structure suffices, as it did for CNS-1's predecessor, the RAP.

The number of columns per processor should be an odd multiple of 32 in order to achieve highest performance. Matrices with short rows will be striped across the rows and columns so that there are at least 32 elements per row and processor. With this arrangement, a reduce-add step using the network would be required for matrix transpose multiplication. The communication pattern maps conveniently to a multiple ring topology, which is easily realized with the CNS-1 network.

Each Torrent has a vector unit with registers of 32 elements each, along with 8 parallel multiply and 8 parallel add pipelines. A wide memory port is optimized for unit-stride accesses. Matrices are striped in memory so that successive columns are in place for successive vector units; in this way memory accesses and vector operations can proceed in parallel among each gang of 32 values.

Pseudocode (not showing all possible optimizations) for multiplying an $n \times m$ matrix M of 16b weights by an $m \times 1$ vector V of 8b activations follows. Rt and rd refer to the abstract use of one or more vector or scalar registers, respectively. The output is placed in V2.

```
for i:=0 to n-1 step 32

    -- initialize sum vector
    Rt[0:31]:=0;

    -- compute 32 elements of the result vector
    for j:=0 to m-1 step 4
       -- read in 4 activations (one 4B memory transfer)
       rd[0:3]:=V[j:j+3];

       for k:=0 to 3
          -- read and mul-add 32 16b weights in vector registers
          -- (one 64B memory transfer)
          Rt[0:31]  += rd[k] * M[j+k,i:j+k,i+31];
       end;
    end;

    -- save result in RDRAM (128B transfer)
    V2[i:i+31]:=Rt[0:31];
end;
```

The code operates by accessing the matrix M in stripes of 32 columns. One row of such a stripe is loaded at a time and is multiplied by the same element of the vector V. The products are accumulated for each column. In the outer loop, the result vector is saved into the RDRAM and the execution switches to the next stripe of 32 columns. The vector V is accessed in blocks of four elements. This code requires about

$$(n/32) \times (m/4) + (n/32 \times m) + (n/32) \approx n \times m/25$$

(for large m) memory accesses. In the final version, the loops would be unrolled and software pipelined to improve performance.

3.13.2 Sparse Connectivity Patterns

It is a goal of CNS-1 to be able to efficiently compute not only dense, fully-connected networks, but also nets with sparse patterns of connectivity. Some of these patterns, such as convolution for image processing, are common and their embedding in our interprocessor network is well understood. Other patterns require some cleverness to preserve performance, by relying on implicit structure. There are several such techniques, and they may be combined.

The simplest and most effective technique is to rely on training set parallelism, not network parallelism. For example, it is sometimes possible to have a local copy of the net for each processor and broadcast training data to each vector unit in the processor, having each run on a separate element of the training set. This has been used with great success for other parallel machines because it eliminates all dependencies between processing units aside from the single reduction occurring once per batch update. This independence allows sparse computations to take place with virtually no penalty for multiprocessing, because the parallelism of the hardware only need only reflect the parallelism of the training set and does not have to conform to implicit parallelism in the network structure, which is difficult when the net has little structure. Unfortunately, this kind of parallelism is not appropriate for all applications.

Nets can often be partitioned into densely connected subcomponents. For example, networks are often described with two or more fully connected layers. Depending on the size of each component and data dependencies, these can be computed serially, using the entire machine for each component, or by subdividing the machine and computing components concurrently. The CNS-1 is naturally divisible into independent rings. Knowledge guiding data allocation is obtained by explicit directives from the developer, heuristics relying on application structure, and run-time statistics.

It is possible to capitalize on special knowledge about the connectivity pattern. What appears to be a general sparse connectivity with weight sharing may actually just be an implementation of convolution, allowing a carefully tuned convolution routine to be used, potentially gaining several orders of magnitude in performance. Full and bus connectivity can be seen as an instances of this.

Sparse codings may be used. It is possible to encode weights not only positionally, but with explicit indices as well; this means that zeroed weights take no space. However, this may not allow efficient use of the vector units because units may not be ordered such that nonzero weights are adjacent. It is possible to change to row and column ordering of the matrix to optimize the use of the vector units.

A theoretical application described in Section 1.3.3 is to be able to support a million nodes with an average of a thousand links each and feed-forward in real time. We assume the worst case, that connections are completely random and no advantage can be made of the vector parallelism. Consider a sparse encoding representing the fan-out for every activation stored at each processor, for all units connected to that activation which reside on the processor. This will require 2B per index, because there are around seven thousand units computed at each processor. Three bytes will be required per activation (the 1B value along with a 2B weight table size). With 16b weights, that amounts to 31 MB per processor to store 7 million weights for 7 thousand units, leaving around 1 MB for code and other variables. The computed partial sums (7000×4B) will fit in the RDRAM cache, and use can be made of the Torrent data cache by first sorting into order the fan-out units from each activation, then sorting into order the activations to maximize temporal and spatial locality. This arrangement allows most accesses to partial sums without even requiring RDRAM access. With reasonable cache behavior we can expect this case to run somewhere around 20 times slower than dense forward propagation.

Another strategy is to pack groups of activation values which are used multiple times, allowing fast vector math on the packed form. An example of where this would be useful is when a layer is connected to a number of units with the same connection strategy. It is unknown at this time if these more exotic techniques will be needed for actual applications.

3.13.3 CNSim

The above sections present details of how some typical problems can be mapped to the CNS-1, but do not specify what the user must write in order for this to happen. At the least, we will have libraries of pre-specified networks and models that have been efficiently coded by wizards. The C++ and Sather language support will enable programmers to make larger systems by writing interfaces between these pre-specified and parameterized library classes. We also plan, however, to provide a much higher-level system to help users construct and experiment with connectionist networks.

We refer to this effort to build an efficient, general purpose connectionist simulator as CNSim. It will combine three ongoing efforts in the group, CLONES, BOB and ICSIM. CLONES is a specialized, but highly efficient simulator that has been running for about two years on the RAP. BOB represents a rewriting of CLONES to improve modularity and present an improved user interface. ICSIM is an evolving, Sather-based general purpose simulator with both RAP and workstation implementations, and is current being ported to the CM-5. The CNSim simulator will allow new high-level objects (such as groups of units) to be built by composition of features and interconnections with other objects. A graphical user interface will make construction by interconnecting these building blocks more interactive and intuitive for the user. In the finished form of CNSim, the user may never realize that the CNS-1 is a parallel machine. All parallelism and data distribution is handled by the class libraries used by the application programs. This is clearly a research project, but we have made significant progress and expect to have an effective software system ready for the CNS-1 hardware.

References

[Asa93a] K. Asanović.
Torrent Architecture Manual, Internal document. International Computer Science Institute and UC Berkeley, 1993

[Asa93b] K. Asanović.
T0 Reference Manual, Internal document. International Computer Science Institute and UC Berkeley, 1993

[Cal93] T. Callahan.
Network Interface Manual, Internal document. UC Berkeley and International Computer Science Institute, 1993.

[CSS+91] D. Culler, A. Sah, K. E. Schauser, T. von Eicken, and J. Wawrzynek.
Fine-grain Parallelism with Minimal Hardware Support: A Compiler-Controlled Threaded Abstract Machine. *Proc. Architectural Support for Programming Languages and Operating Systems*, pages 164—175, 1991.

[Dal90] W. J. Dally.
Virtual-Channel Flow Control. *Proc. IEEE 17th Annual International Symposium on Computer Architecture*, pages 60—68, 1990.

[DS86] W. J. Dally and C. Seitz.
The torus routing chip. *Journal of Distributed Computing*, I(3):187—196, 1986.

[DW89] W. J. Dally and D. S. Wills.
Universal Mechanisms for Concurrency. PARLE-89, pages 19—33, Goos and Hartmanis, eds., Springer-Verlag, 1989.

[FLR92] J. Feldman, C-C. Lim and T. Rauber.
The Shared-Memory Language pSather on a Distributed-Memory Multiprocessor. *Second Workshop on Languages, Compilers and Run-Time Environments for Distributed-Memory Multiprocessors,* Boulder, CO, 1992.

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

[IYM+89] A. Iwata, Y. Yoshida, S. Matsuda, Y. Sato, and N. Suzumura.
An Artificial Neural Network Accelerator using General Purpose Floating Point Digital Signal Processors. *Proc. International Joint Conference on Neural Networks,* pages II:171—175, 1989.

[Lyo92] R. Lyon.
Analog Implementations of Auditory Models. *Proc. of Fourth DARPA Workshop on Speech and Natural Language*, Asilomar, CA, In Press.

[MBLD92] O. Matan, C. Burges, Y. Le Cun, and J. Denker.
Multi-Digit Recognition Using a Space Displacement Neural Network. *Proc. of Fourth Annual Conference on Neural Information Processing Systems - Natural and Synthetic*, Denver, CO, pages 488—503, 1992.

[MBK+92] 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 Computing*, Special Issue on Neural Networks, XIV:248—259, 1992.

[MB90] N. Morgan and H. Bourlard.
Continuous Speech Recognition Using Multilayer Perceptrons with Hidden Markov models. *Proc. IEEE International Conference on Acoustics, Speech, & Signal Processing*, Albuquerque, NM, pages 413—416, 1990.

[MC91] Y. K. Muthusamy and R. A. Cole.
A segment-based automatic language identification system.
Proc. of Fourth Annual Conference on Neural Information Processing Systems - Natural and Synthetic, Denver, CO, pages 241—248, 1992.

[MP89] J. Malik and P. Perona.
A Computational Model of Texture Segmentation. *IEEE Conference on Computer Vision and Pattern Recognition*, San Diego, CA, pages 326—332, June 1989.

[Omo91] S. Omohundro.
The Sather Language. International Computer Science Institute, Berkeley, CA, 1991.

[PS81] D. Patterson and C. Sequin.
RISC I: A Reduced Instruction Set VLSI Computer. *Proc. of 8th Annual Symposium on Computer Architecture*, Minneapolis, MI, pages 443—457, May 1981.

[Ram92] Rambus Technology Guide, Preliminary Edition.
Rambus Incorporated, 2465 Latham Street, Mountain View, CA, 1992.

[RBR+91] U. Ramacher, J. Beichter, W. Raab, J. Anlauf, N. Brüls, U. Hachmann, and M. Wesseling.
Design of a 1st Generation Neurocomputer. *VLSI Design of Neural Networks*, Kluwer Academic, pages 271—310, 1991.

[RMCF92] S. Renals, N. Morgan, M. Cohen, and H. Franco.
Connectionist Probability Estimation in the DECIPHER Speech Recognition System. *Proc. IEEE International Conference on Acoustics, Speech, Signal Processing*, pages 601—604, San Francisco, CA, 1992.

[Sch90] H. Schmidt.
ICSIM: Initial Design of an Object-Oriented Net Simulator. *International Computer Science Institute Technical Report*, TR-90-055.

[Sha88] L. Shastri.
A Connectionist Approach to Knowledge Representation and Limited Inference. *Cognitive Science*, Vol. 12, No. 3, pages 331—392, 1988.

[Vig70] S. Viglione.
Applications of Pattern Recognition Technology. *Adaptive, Learning and Pattern Recognition Systems*, J. Mendel and K. Fu eds., Academic Press, 1970.

[vEC+92] T. von Eicken, D. Culler, S. Goldstein, and K. Schauser.
Active Messages: a Mechanism for Integrated Communication and Computation. *Proc. of 19th International Symposium on Computer Architecture*, pages 256—266, 1992.

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

Index

Table of Contents

Table of Figures