INTERNATIONAL COMPUTER SCIENCE INSTITUTE

1947 Center Street • Suite 600 • Berkeley, California 94704 • 1-510-642-4274 • FAX 1-510-643-7684

All-to-all Broadcast on the CNS-1*

Silvia M. Müller†

TR-93-082

December 1993

Abstract

This study deals with the all-to-all broadcast on the CNS-1. We determine a lower bound for the run time and present an algorithm meeting this bound. Since this study points out a bottleneck in the network interface, we also analyze the performance of alternative interface designs. Our analyses are based on a run time model of the network.

*The CNS-1 (Connectionist Network Supercomputer) project is a collaboration of the University of California at Berkeley and the International Computer Science Institute

†ICSI and CS Department, University of Saarland, Germany. E-mail smueller@icsi.berkeley.edu or smueller@cs.uni-sb.de

1 Introduction

Some neural network algorithms for sparsely connected networks require an all-to-all broadcast described as follows:

> *A vector of n data elements is spread equally over the p nodes of a multicomputer. Each node stores $n_p = n/p$ elements. During an all-to-all broadcast, each processor collects the whole vector.*

In this work, we analyze how fast the CNS can execute such a transfer. We first look at some design principles of the network and the network interface. This gives us a lower bound of the broadcast time. We then present a simple algorithm which meets this bound if the vectors are not too short. These analyses indicate a bottleneck in the network. In section 5 we therefore discuss design alternatives to overcome the bottleneck and to speed up the all-to-all broadcast. And finally, we sketch the potential impact of the broadcast on the performance of the CNS.

Without loss of generality, we assume that a vector element is four bytes. If the elements are only one byte, four elements can be packed into one word. This packing can be done with some loads from and stores to the on-chip data cache.

The CNS as described in [ABC+93, AC93] is still in design. We therefore base our analyses on the run time model developed in [Mül93]. In the next section we sketch the network and its timing, but for more details we refer to [ABC+93, AC93].

2 Network overview

The CNS has a cylindrical topology. This is a mesh with wraparound in only one dimension. Each ring holds 32 nodes. Neighboring nodes are connected by a bidirectional link, with a capacity of 8b[1] per CPU cycle and direction. Figure 1 shows a block diagram of the network interface.

Transfer overhead occurs on the sending and on the receiving side. It is partially related to processor activities and partially to network activities. The processor, for example, executes an interrupt and moves the data into or from special transfer registers. For simplicity, we assume that all overheads require about the same time $ov = 5$. That is a realistic value as Tim Callahan showed in his recent analysis ([Cal93]). According to the timing model ([Mül93], p. 5-7), it then takes $4\,ov + 1 + m_{length}$ cycles to transfer a message of m_{length} bytes between neighboring nodes. Each message includes nine header bytes.

The sender copies the message into a buffer of its network interface. The network interface then sends the buffer message byte by byte through the network. The first byte leaves the buffer $2\,ov + 1$ cycles after the start of the transfer, and the last byte leaves it $m_{length} - 1$ cycles later. We assume that the node can start a new conflict free transfer in the same direction $m_{length} + 1$ cycles after it started the previous transfer. This message arrives in the network buffer $2\,ov$ cycles later. So, only the overhead can be hidden, when sending two messages over the same link.

[1] **b** designates *bit* and **B** designates *bytes*

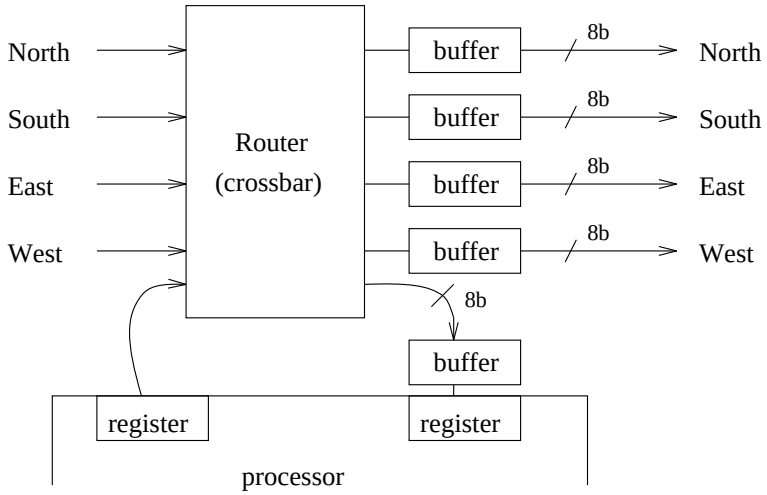

Figure 1: Data network interface

3 Lower bound

A node could receive up to four bytes per cycle, one byte from each of its four links, but there is only one buffer in a network node forwarding data to the processor. The data are copied to a receiver register after the message has completely arrived in the buffer. Then the buffer is free for other messages. At best, each node can receive messages of m_{length} bytes every $m_{length}+1$ cycles. That already gives us a lower bound for the broadcast time on the CNS.

Each node sends n_p vector elements and receives $n_p(p-1)$ elements. A message can transfer only one vector register. The length of vector registers varies between 1 and the maximal length vlr, which usually is 32. So n_p vector elements are transferred in $x = \lceil n_p/vlr \rceil$ chunks and each node receives at least $anz = x\,(p-1)$ messages with $m_e = \lceil n_p/x \rceil \geq vlr/2+1$ elements each.

At best — assuming no blockings in the network or in the nodes, the nodes receive the first byte $2\,ov+2$ cycles after starting the broadcast. Every $m_{length}+1$ cycles, the nodes receive the first byte of the next message. The first byte of the last message arrives after $2\,ov+2+(m_{length}+1)\,(anz-1)$ cycles, the last byte arrives $m_{length}-1$ cycles later. It takes the receiving node additional $2\,ov$ cycles to extract the message from the network. Consequently, the all-to-all broadcast requires at least $T(p,n_p)$ cycles on the CNS,

$$\begin{aligned} T(p,n_p) &\equiv 2\,ov+(m_{length}+1)\,anz+2\,ov \\ &= 4\,ov+(m_{length}+1)\,(p-1)\left\lceil\frac{n_p}{vlr}\right\rceil. \end{aligned}$$

This bound does not take into account, whether the node and the network can handle the data fast enough.

4 All-to-all broadcast algorithm

The following broadcast algorithm meets the lower run time bound $T(p, n_p)$, if the vector has more than $p \cdot (vlr + 1)$ elements and if the broadcast is not delayed by computation or other transfers.

The nodes are arranged in a directed ring. So, each node only sends to its successor in the ring and only receives from its predecessor. After transferring its own part of the vector, each node forwards the data received from the preceding node. The all-to-all broadcast is finished after each node received $p - 1$ parts of the vector. Sends occur every $m_{length} + 1$ cycles. The data are transferred in chunks of m_e elements.

4.1 Mapping a ring into the CNS topology

There are many ways to map a directed ring into a cylinder, using only nearest neighbors. We chose to connect the nodes along the columns, alternating the direction per column. Connecting the columns by horizontal links on the top or bottom yields a directed ring. Figure 2 illustrates this mapping for a 8×4 cylinder.

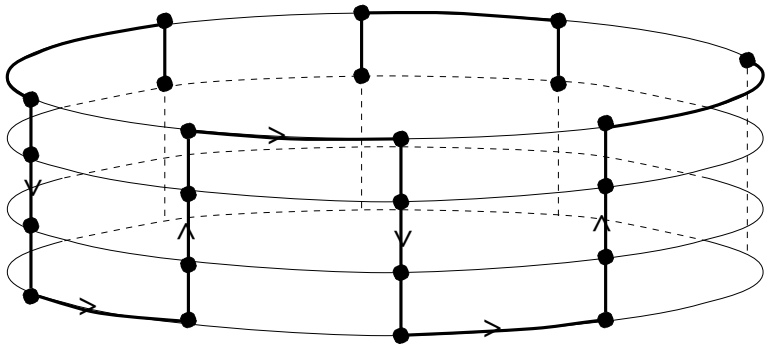

Figure 2: Mapping a ring into a 8×4 cylinder

4.2 Run time analysis

We prove the following facts in order to show that our algorithm holds the time bound $T(p, n_p)$:

(1) sending and receiving overheads do not collide, when scheduling messages every $m_{length} + 1$ cycles, and there occur no blockings in the network,

(2) the memory bandwidth is sufficient to read and write m_e vector elements every $m_{length} + 1$ cycles, and

(3) data arrive early enough, so that messages can be send on time (every $m_{length} + 1$ cycles).

These facts guaranty that the nodes send and forward the messages every $m_{length} + 1$ cycles, that the messages arrive in the same interval, and that the nodes can consume the arriving data. So, all the assumptions we implicitly made when proving the time bound are met.

(1) No collision

Transfers are scheduled every $m_{length} + 1$ cycles, that is $4m_e + 9 + 1 \geq 78$ cycles. Send and receive overheads only require $2\,ov = 10cc$ (CPU cycles) each, and succeeding send or receive overheads will therefore not collide. Receives occur $2\,ov + m_{length} + 1$ cycles after the corresponding sends, so they just start after the execution of the next send overhead.

The mapping guaranties that each link in the CNS network supports at most one link of the ring. A link needs $m_{length} + 1$ cycles to transfer the message. Consequently, there also occur no blockings in the network.

(2) Memory bandwidth

The CNS has a multilevel memory hierarchy. In the worst case, it takes 28 cycles to load vlr words into the CPU and 23 cycles to write them back. The values are based on the memory model presented in [Mül93]. Message overhead and memory accesses can largely be overlapped. After scheduling the load (store), the processor can execute the receive (send). Message overhead and memory accesses therefore keep the processor busy for no more than $\max(28 + 23, 21) = 51$ cycles. New data only arrive every $m_{length} + 1$ cycles, which is at least 78 cycles.

(3) On time data arrival

Each node sends its part of the vector in chunks of m_e elements. These $x = \lceil n_p/vlr \rceil$ messages get started every $m_{length} + 1$ cycles, and as a consequence of (1) they also arrive in the same interval. If the first message arrives early enough to be forwarded on time, then the following $x - 1$ also do not cause a problem and the next forwarding round also starts on time. As a consequence, all messages get started in the required time pattern.

Let us assume that the broadcast starts at time 0. The first message then arrives at time $(4\,ov + (m_{length} + 1))cc$ and should be forwarded at time $x\,(m_{length} + 1)cc$. That is no problem if x is at least 2, because $m_{length} + 1 \geq 78 \geq 4\,ov$.

Short vectors

For a total vector length of $vlr \times p$ words or less, the nodes can transfer their data in one message. This causes problems in the forwarding step. The nodes cannot start the forwarding before they have completely finished the previous transfer. New sends start now every $4\,ov + m_{length} + 1$ cycles, and the all-to-all transfer requires $(4\,ov + m_{length} + 1)(p - 1)$ cycles.

5 Design impact

Compared with theoretical results on the performance of all-to-all broadcast ([FL91]), the CNS shows the run time behavior of a *processor-bound*[2] multicomputer. The network nodes have a maximal degree of $\Delta = 4$, and so a *link-bound*[3] version of the CNS could reduce the all-to-all broadcast time upto one fourth. The transfer would still take at least $T_{th}(p, n_p)$ cycles:

$$T_{th}(p, n_p) = \frac{m_{length} + 1}{\Delta} \cdot (p - 1) \cdot \left\lceil \frac{n_p}{vlr} \right\rceil .$$

We now analyze, what causes the bottleneck and how to overcome it.

5.1 Location of the bottleneck

Figure 3 shows the interconnection of the network node and of the main processor parts. The bandwidths of the links and busses are listed in table 1. These numbers show, that the processor when working from data cache or register file could serve upto four network links. When accessing random data in the main memory, the processor could still serve two links. So the bottleneck is neither the processor nor the memory system but the network interface. A link-bound version of the CNS is hardly possible, at least for general access patterns in main memory, but a *two-port* transfer is realistic. To do so, the bandwidth of the network interface has to be increased. This can be achieved by more or faster links between processors and network nodes.

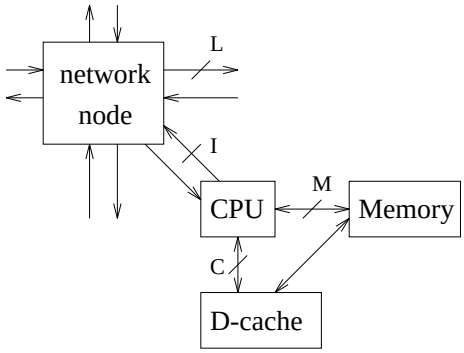

Figure 3: Data network interface

5.2 Faster network interface

The interface can be sped up by doubling the number of links between processor and network node. That would increase the critical bandwidth to 2 I. A slight variation of the original broadcast algorithm then achieves twice the original broadcast performance (at least for large vectors).

[2] each processor can only use one link at a time

[3] each node can use all its links at the same time

	Links		Memory			Vector Unit	
	L	I	C	M		1 Operation	2 Operations
				max	min	1 byte	4 bytes
read	1	1	32	9	4	16	128
write	1	1	32	12	5	8	64
read & write	yes		no			yes	

Table 1: Bandwidth of network links and busses ([B/cc]). The table also shows, whether a port supports reads and writes at the same time.

Modified algorithm

A ring can be traversed in positive and negative direction. N^+ indicates the neighbor of node N in positive direction and N^- the neighbor in negative direction (figure 4).

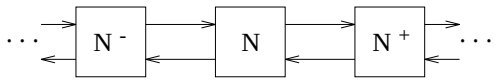

Figure 4: Node arrangement

The nodes are still arranged in a ring, but they now initiate the all-to-all broadcast by sending their data to their two neighbors N^+ and N^-, starting with N^+. In the second phase, the nodes forward data from neighbor N^- to neighbor N^+ and vice versa. Messages are transported $\lfloor p/2 \rfloor$ hops in positive and $\lceil p/2 \rceil - 1$ hops in negative direction. Sends in the same direction still start every $m_{length} + 1$ cycles, sends in negative direction start $4\,ov$ cycles later than the corresponding transfers in positive direction.

Run time

According to the bandwidth comparison in the previous section, the CPU and the memory system can deal with twice as much data as in the original algorithms. The processors have $m_{length} + 1 \geq 78$ cycles to execute the overhead and the memory accesses of two transfers. The overhead requires $8\,ov = 40$ cycles. Since memory accesses and overhead instructions can be partially overlapped, efficient scheduling is all that is needed to avoid collisions.

We saw in the original algorithm that the receive overhead follows the send overhead of the next transfer. Both overheads together require $4\,ov$ cycles. We therefore schedule the transfers in negative direction $4\,ov$ cycles after the corresponding transfers in positive direction.

For large vectors ($n_p > vlr$) and an even number of processors, the all-to-all broadcast

now requires $T_{2I}(p, n_p)$ cycles,

$$T_{2I}(p, n_p) = 4\,ov + (m_{length} + 1)\,\frac{p}{2}\,\left\lceil \frac{n_p}{vlr} \right\rceil .$$

A faster link to the processor

Another way to achieve a bandwidth of 2 I between the processor and the network node, is to make that link twice as fast. The all-to-all broadcast algorithm stays basically the same, but the interval between messages changes. It now takes the processor $y = \lceil m_{length}/2 \rceil + 1$ cycles to transfer the message to the network node. Consequently, the processor starts new transfers every y cycles in alternating directions. This time $y \geq 40$ is still sufficient to execute $4\,ov = 20$ cycles overhead. The all-to-all broadcast now requires $T'_{2I}(p, n_p)$ cycles,

$$T'_{2I}(p, n_p) = 4\,ov + \left(\left\lceil \frac{m_{length}}{2} \right\rceil + 1 \right)\,(p-1)\,\left\lceil \frac{n_p}{vlr} \right\rceil .$$

5.3 Faster network links

An alternative solution is to speed up all network links, including the connection between processor and network, by the same factor. With this change, the memory system will become the bottleneck. The design team is currently trying to achieve a speedup factor of two. Bandwidths of 2 L and 2 I would also double the all-to-all broadcast performance, even using the original algorithm.

As in the previous case, the links only need $\lceil m_{length}/2 \rceil$ cycles to transfer a message, and new messages can be sent or forwarded every $\lceil m_{length}/2 \rceil + 1$ cycles. For large vectors ($n_p > vlr$), the all-to-all broadcast then requires $T_{2I,2L}(p, n_p)$ cycles,

$$T_{2I,2L}(p, n_p) = 4\,ov + \left(\left\lceil \frac{m_{length}}{2} \right\rceil + 1 \right)\,(p-1)\,\left\lceil \frac{n_p}{vlr} \right\rceil .$$

So, speeding up all network links or only the link to the processor has the same effect on the all-to-all broadcast time. However, for other transfers faster network links might be useful.

5.4 Restrictions in the protocol

We assumed that a node can partially overlap transfers even when they use the same link. However, the network protocol may not support this. In the current design, an acknowledge signal will be sent back at the end of the transfer. With a one-bit signal, a node could probably only overlap transfers using different links. The nodes then always have to use more than one link in order to achieve a high all-to-all broadcast performance.

6 Performance impact

We now analyze how the broadcast time influences the performance of the CNS, when evaluating the activations in a sparse neural network with half a million units having an average of 500 connections per unit.

Communication

In the parallelization described in [GM94], the weights and activations are equally spread over the p processors but the processors need all activations for their computation. The activations are only one byte wide, and so four of them can be packed into one memory word. The all-to-all broadcast then requires

$$T = 20cc + (p-1)\left\lceil \frac{5 \cdot 10^5}{p\ 4\ vlr} \right\rceil\ 138$$

cycles with a cycle time of 8 ns. Between two transfers, the CPU can spend at most $m_{length} + 1 - 2\ ov \leq 128$ cycles executing other code. During one all-to-all broadcast that adds up to about half a million cycles per node, that is almost 93% of the whole transfer time. At best, this time can completely be used for the computation of the next iteration.

Computation

It essentially takes one multiply-accumulate operation to evaluate a connection, but due to the sparseness of the matrix there also occur address calculations and indexed memory accesses. The address calculations need one addition per connection, and the indexed memory accesses might also require considerable amount of run time.

A node has a peak performance of 8 multiply-accumulate operations per cycle. The computation of one iteration therefore requires at least

$$C_{min} = \frac{2 \cdot 2.5 \cdot 10^8}{8p} = \frac{5 \cdot 10^8}{8p}$$

cycles, when executing it at peak performance. That does not include time for main memory accesses and so it is a very optimistic lower run time bound for the computation. On the other hand, the analysis in [GM94] gives an upper bound of

$$C_{max} = \left\lceil \frac{5 \cdot 10^8}{p} \right\rceil 6$$

cycles. This run time is 24 times slower than the best case, but it is very difficult to vectorize the indexed memory accesses, the most timeconsuming part of the code. That already extends the run time by a factor 16. Another factor of 1.5 gets lost, because the algorithm often accesses only one byte per memory page.

Performance comparison

Table 2 lists execution time and performance of one iteration of our benchmark task on the CNS. The table also shows three different computation times: C_{min}, C_{max} and the maximum time a CPU use for other operations during the all-to-all broadcast. The total run time is based on C_{min}. We assume that computation and transfer can completely be overlapped.

The all-to-all broadcast has a 2 to 36% impact on the total run time, if the computation time of the optimized sparse case code is close to C_{max}. In the best case, the computation

Band-width	Nodes	All-to-all Broadcast	Computation Time			Total	Performance	$\frac{\text{Total}}{C_{min}}$
			available	C_{min}	C_{max}	min	(C_{min})	
I, L	128	4.35	4.03	3.91	94	4.35	58	1.1
	256	4.50	4.18	1.95	47	4.50	56	2.3
	512	4.51	4.19	0.98	23	4.51	55	4.6
	1024	4.52	4.19	0.49	12	4.52	55	9.3
2I, 2L	128	2.20	1.89	3.91	94	4.22	59	1.1
	256	2.28	1.96	1.95	47	2.28	109	1.2
	512	2.29	1.96	0.98	23	2.29	109	2.3
	1024	2.29	1.96	0.49	12	2.29	109	4.7
Unit		[ms]					[G CPS]	

Table 2: Run time of one iteration of the benchmark task on different versions of the CNS

can be executed at peak performance and can completely be overlapped with the transfer. The all-to-all broadcast then dominates the run time. On the 1024-node machine, the execution of this task then takes at least 3.7 times longer than the whole computation time. The computation time C_{min} does not include memory access times. Including memory access times, the computation will hardly be faster than $8\,C_{min}$, at least with the RDRAM memory system presented in [ABC+93, AC93]. The total run time then adds up to about $12\,C_{min}$.

7 Conclusion

The bandwidth between the processor and the network is the same as the bandwidth between two network nodes. That results in a processor-bound CNS. The network interface is a bottleneck, but it can be speeded up by a factor of two before other hardware components limit the performance of the all-to-all broadcast.

All-to-all broadcast has an considerable performance impact but does not cause disastrous performance loss, at least not on our benchmark task. The CNS can run all-to-all broadcast almost without any visible transfer overhead, and the CPUs can use upto 93% of the transfer time for other computations.

References

[ABC+93] K. Asanović, J. Beck, T. Callahan, J. Feldman, B. Irissou, B. Kingsbury, P. Kohn, J. Lazzaro, N. Morgan, D. Stoutamire, and J. Wawrzynek. CNS-1 architecture specification. Technical Report TR-93-021, International Computer Science Institute and UC Berkeley, 1993.

[AC93] K. Asanović and T. Callahan. *Torrent Architecture Manual.* International Computer Science Institute and UC Berkeley, 1993. Internal document, revisions 1.5/1.9.

[Cal93] T. Callahan. *CNS-1 Networks*, October 1993. Talk at the International Computer Science Institute, Berkeley CA.

[FL91] P. Fraigniaud and E. Lazard. *Methods and Problems of Communication in Usual Networks*. Ecole Normale Supérieure de Lyon, France, 1991. Internal document, October.

[GM94] B. A. Gomes and S. M. Müller. A performance analysis of CNS on sparse connectionist networks. Technical report, International Computer Science Institute and UC Berkeley, 1993/94.

[Mül93] S. M. Müller. A performance analysis of the CNS-1 on large, dense backpropagation networks. Technical Report TR-93-046, International Computer Science Institute, Berkeley, 1993.