Preliminary basic performance analysis of the cedar multiprocessor memory system

K. Gallivan, W. Jalby, S. Turner, A. Veidenbaum
H. Wijshoff

RUU-CS-91-15
June 1991

Utrecht University
Department of Computer Science
Padualaan 14, P.O. Box 80.089,
3508 TB Utrecht, The Netherlands,
Tel. : ... + 31 - 30 - 531454

Preliminary basic performance analysis of the cedar multiprocessor memory system

K. Gallivan, W. Jalby, S. Turner, A. Veidenbaum

H. Wijshoff

Technical Report RUU-CS-91-15
June 1991

Department of Computer Science
Utrecht University
P.O.Box 80.089
3508 TB Utrecht
The Netherlands

ISSN-0924-3275

Preliminary Basic Performance Analysis of the Cedar Multiprocessor Memory System*

K. Gallivan† W. Jalby†† S. Turner† A. Veidenbaum†
H. Wijshoff‡

† Center for Supercomputing Research and Development,
University of Illinois at Urbana-Champaign, USA
†† IRISA, University of Rennes, France
‡ Department of Computer Science, Utrecht University,
the Netherlands

Abstract

In this paper we present some preliminary basic results on the performance of the Cedar multiprocessor memory system. Empirical results are presented and used to calibrate a memory system simulator which is then used to discuss the scalability of the system.

1 Introduction

The Cedar system is a multivector processor comprising 4 clusters of 8 vector computational elements (CE's) and a global memory system. Each cluster is a modified Alliant FX/8 in which the 8 CE's share a data cache and a cluster memory system. The 32 CE's are connected to 32 global memory banks by a two-stage Ω-network whose basic component is an 8-by-8 crossbar and whose interconnection topology is shown in Figure 2. Note the 32-to-16 fan-in followed by a 16-to-32 fan-out connection scheme. The details of the system are given in a companion paper [4], but we recapitulate some of the aspects of interest in this paper.

One of the most striking aspects of the architecture is the memory system. Each cluster has a private hierarchy consisting of vector registers private to each CE, a shared data cache and a shared cluster memory. Intercluster communication is accomplished via the large shared global memory accessible by all of the CE's. Each CE communicates with the global network via a private global interface board (GIB) which contains a vector prefetch unit that can be used to access elements independently of CE processing and thereby mitigate the cost of global memory latency. The prefetch unit is capable of issuing a block of addresses given a base address, a stride and a length. The prefetch block size can be up to 512 64-bit words. When elements of the prefetch block return to the GIB they are

*This work was supported by the Department of Energy under Grant No. DE-FG02-85ER25001, the National Science Foundation under Grant No. NSF 89-20891, the NASA Ames Research Center under Grant No. NASA NCC 2-559, Cray Research Inc. and Alliant Computer Systems.

placed in the prefetch buffer which is essentially a direct map cache with some restrictions. As is seen below, the exploitation of the cache nature of the prefetch buffer can improve global memory performance significantly.

Clearly, exploiting this memory system efficiently is crucial if a reasonable fraction of the peak performance of Cedar is to be achieved in practice. This paper presents some of the results of a global memory performance characterization effort presently underway. It includes the use of a simulator, which has been calibrated based on empirical observations of Cedar memory performance, to investigate the scalability of the approach.

2 Memory system experiments

2.1 Description of experiments

2.1.1 The LOAD/STORE kernels

The memory system has been explored by a generalization to the Cedar architecture of earlier work done within a cluster for the purposes of characterization and performance prediction, [1, 2]. This approach makes use of a set of parameterized memory access kernels informally referred to as the LOAD/STORE kernels. In order to predict performance on Cedar these memory system kernels must be augmented with a similar set of kernels which isolate the effect of the control constructs available on Cedar. In this paper, however, we will only give a brief summary of some of the basic results for the LOAD/STORE kernels. A more complete description of the LOAD/STORE characterization of Cedar and the associated performance prediction will be discussed in a forthcoming paper.

The kernels were built in order to investigate the behavior of the memory system stressed by various memory request streams. They were parameterized in a manner which respects the following constraints:

- It must be possible to adjust the set of parameters in such a way to emulate memory requests sequences arising from real codes.
- The parameters have to be chosen so they can be varied independently in order to analyze precisely the behavior of the memory system and its interaction with the memory request stream.

Our study is restricted to a steady state analysis, i.e., each CE loops around the same piece of code (which by construction will have exactly the same pattern of memory requests) a large number of times. The main reasons for concentrating on steady state analysis were to limit the number of possible parameters affecting the behavior and to reduce the number of cases which are pathologically difficult to analyze. There are ways to approximate the effect of transient behavior on Cedar for performance prediction but they are beyond the scope of this paper.

The main parameters that were varied during the experiments are:

1. Number of CE's and Clusters
2. Mode of request on each CE: scalar, vector without prefetch, vector block prefetch (implicit and explicit)

3. Type and pattern of requests: various LOAD, STORE combinations such as, LOAD, STORE, LOAD-STORE, LOAD-LOAD-STORE, etc.

4. Temporal distribution of requests

5. Spatial distribution of requests: stride and offset

6. Scheduling

Let us examine in turn each of these parameters and their potential effect on the memory system behavior.

The number of CE's and Clusters issuing requests is the most obvious parameter to vary the workload imposed on the memory. However, it should be noted, that a priori for a same number of CE's requesting, the partitioning of the active CE's across the clusters may have a nonnegligible impact. For example given 8 CE's active, the behavior maybe different if the 8 CE's belong to the same cluster or if they are evenly distributed across the clusters.

The mode of request obviously affects the issue rate of the requests but it also alters the way they are handled. In scalar mode and vector mode without prefetch, the processor can have at most 2 outstanding requests to global memory pending at any time. Since vector mode implies that a series of independent fetches are required 2 outstanding requests are maintained for the duration of the vector instruction. In scalar mode, however, the number of outstanding requests maintained over a series of instructions depends upon resource dependences. Each request may have to pay the full cost of latency.

When prefetch is used the prefetch unit can issue and handle several outstanding requests to global memory. The number of outstanding requests allowed by the prefetch unit is controlled by length of the prefetch block, *bpl*, which can range from 1 byte to 512 64-bit words. The prefetch unit can be used in two modes: implicit and explicit. In both modes, a burst of *bpl* requests is emitted by the prefetch unit at a rate of one request per cycle. In implicit mode for a LOAD, the CE then attempts to transfer elements, in order, from the prefetch buffer into vector registers. If the next data element in order has not arrived in the prefetch buffer the CE stalls until the data is available. (Data returning from global memory, however, is loaded into the prefetch buffer any order by the prefetch unit.) The implicit mode, therefore, is effectively an accelerated global memory vector load instruction.

In explicit mode, the CE is allowed to continue executing any instructions following the prefetch start which does not involve the prefetch unit, e.g., operations involving registers or cluster memory. At some point, however, the CE will attempt to access the data that was prefetched and it will then behave in a manner identical to the implicit prefetch.

The type of request, LOAD or STORE, affects the significance of the mode of request. In the case of loads, the discussion above applies. On the other hand, requests for writes are emitted as fast as the processing of the particular instruction allows, e.g., every cycle for a vector write. The CE does not wait for an acknowledgment of a write completion. The GIB/CE pair has an explicit instruction, that is used for synchronization purposes, which stalls the CE until all outstanding write requests for the CE have completed. In this paper, we will will concentrate LOAD requests.

The pattern of memory request is intended to study the effect of interleaving vector LOADS and STORES. The reason for studying the mixing the types of requests is that they result in different traffic patterns on the forward and reverse networks. In the case of a LOAD (STORE) request, a packet of one word (two words) traverses the forward network and a packet of two words (one word) travels back from memory across the reverse network. This asymmetry may generate a difference in performance between a long sequence of LOADS followed by a sequence of STORES and a sequence interleaving LOADS and STORES. The first sequence heavily loads the reverse network initially, during the LOAD sequence, then the forward network, during the STORE sequence. The second sequence achieves a better temporal balance on both networks.

The temporal distribution mainly refers to the variation in issue request rate. This is achieved in two ways. Inside a block prefetch request (implicit or explicit), the use of different vector mask values allows us to emulate various distributions. For example, a mask set to the value 010101 will in be in fact equivalent to issuing a request every other cycle. More complex patterns allow the generation small bursts of requests.such as 11110000 The insertion of a variable number of null operations, NOPS, between the prefetch blocks allows us to vary the distribution at a higher level. (This level is more useful from a performance prediction point of view).

The spatial distribution essentially covers the way the banks are addressed. The simplest parameter is the stride. It affects the order in which the memory banks are accessed as well as the number of distinct memory banks accessed. For example, assuming that every processor starts in the same bank (0), striding by 2 will concentrate all the requests to the even numbered banks. Another parameter used to affect the spatial distribution is the offset. This parameter selects the bank in which each processor starts its requests and is typically a function of the processor number, i.e., it depends upon which cluster a CE is in and its local CE number $0 \leq p \leq 7$. As is seen below, the careful selection of these parameters can significantly affect the bandwidth from global memory.

Since our experimental templates use loops as the basic control construct, the iteration scheduling also plays a key role. Two types of scheduling have been studied: self scheduling in which the iterations of the loop are dynamically allocated to each processor, and static scheduling where the iterations assigned to a given processor are determined a priori. In this last scheme, the number of iterations is equally distributed among the processors. Therefore any load imbalance with that scheme will allow to detect asymmetries in the behavior of each processor. This is particularly significant in networks where conflict arbitration is based on processor numbers. The results below are all from tests which used self-scheduling.

2.1.2 Description of a basic LOAD/STORE kernel

The code which implements the vector-concurrent prefetch version of the LOAD kernel is typical. The other forms are simple modifications. (The crucial portions of the kernels are implemented in assembler but are given below in a high-level language.)

The code comprises several nested loops. The outermost loop distribute the work among the clusters: each cluster will run exactly the same code. (This loop has been implemented to minimize as much as possible the associated overhead.) Inside each cluster, a loop over N_i iterations is executed where each iteration consists of a CE performing S_i memory accesses as a series of prefetches with block size S_{bf}. The temporal distribution of access is controlled by the insertion of NOPS at key points in the iteration. The values of the parameters used in the experiments is given in Table 1.

Table 1: Parameters for experiments.

N_i	512
S_i	512
S_{bf}	32, 64, 128, 256, 512
I_{nop}	0, 128, 256, 384, 512, 640, 768, 896, 1024 1536, 2048, 2560, 3072, 3584, 4096

```
        Loop over clusters
            DO $i = 1, N_i$ Parallel loop over iterations
            (self-scheduled within a cluster)
$P1a$:              Prologue (executed only once on each CE)
$P1b$:                  Preparation of the iteration
$P2$:                   Execution of $I_{nop}$ NOPS
                        DO $j = 1, \lceil S_i/S_{bf} \rceil$ Loop over block fetches (sequential on a CE)
$P3$:                       Enabling a block fetch of $S_{bf}$ words
                            DO $k = 1, \lceil S_{bf}/32 \rceil$
$P4$:                           Vector LOAD of 32 elements
                            ENDDO
                        ENDDO
            ENDDO
        End loop
```

Figure 1: Basic LOAD template code.

The code for the basic LOAD kernel is shown in Figure 2.1.2 The loop is decomposed itself into 5 major segments:

1. $P1a$: This code corresponds to the setup of the loop on a cluster. It is executed once by each CE at the beginning of the loop.
2. $P1b$: This phase covers the loading of all the parameters necessary to the execution of an iteration on one CE. It is executed on each iteration.
3. $P2$: this section of code consists of a sequence of I_{nop} NOP instructions to control the temporal distribution.
4. $P3$: The block prefetch mode is enabled here, allowing the CE/GIB to perform memory requests in blocks of S_{bf} words.
5. $P4$: This section of code loads $N_{bf}\lceil S_{bf}/32 \rceil$ blocks into the vector register from the prefetch buffer. It is needed because of the vector register length of 32.

2.2 Basic performance limits

Key considerations in interpreting the results below are the underlying performance limits implied by the components of the memory system and the experimental set up. In this section we summarize these limits.

The CE's within each cluster are vector processors which can, in principle, execute two floating point operations per cycle. (A cycle is 170 ns.) This implies a peak execution rate of 11.76 Mflops per CE. If one takes into account the startup of the chained vector instructions which perform the computations the effective peak rate drops to 8.56 Mflops per CE. In practice, a reasonable rule of thumb is that 55 to 65 Mflops per cluster is the most one can expect. Since each CE has one port to the cluster memory system, a cluster cache is designed to have a hardware limiting access rate of 8 64-bit words per cycle or 47 MW/s. The cluster memory is bandwidth limited in the sense that it cannot match these data demands of the 8 CE's. It has a hardware limiting access rate of 4 words per cycle or 23.5 MW/s. As a result, computations with limited data locality that cannot exploit the cluster memory hierarchy will be bandwidth limited and their performance as a function of the number of processors used will tend to level off after 4 or 5 CE's.

The components of the global memory system are, at least in terms of hardware limits, fairly well balanced with the cluster memory hierarchy. Each memory bank can deliver 1 word every two cycles for a rate of 2.94 MW/s per bank. A network link can transfer two words per cycle. However, since on a read (write) an address/data two-word pair must be transferred in the reverse (forward) network, a link can be thought of as effectively transferring 1 word per cycle or 5.94 MW/s per link. The 16 interstage network links in each direction therefore balance with the 32 memory banks. The GIB/CE pair operating in prefetch mode can issue a request to the network and absorb a returning data word from the network in one cycle for a potential demand of 5.88 MW/s per CE. When prefetch is not used the performance is limited by the number of outstanding requests to memory a CE allows (presently 2). A cost of roundtrip latency of about 13 cycles is paid per pair of words yielding approximately 1 MW/s per CE.

Table 2 summarizes the hardware limits for each of the components of the memory system and the aggregate limit for a 32 CE/memory bank Cedar configuration. Clearly, when all CE's are accessing data only the cluster cache level can satisfy the data demands of the vector functional units. Of course, if a very small number of CE's are accessing data then global memory with prefetch could satisfy the data demand since the GIB is designed to issue requests at the CE peak rate.

Table 3 shows the relative performance degradation of each level of the memory hierarchy with the cluster cache level taken as 1. The hardware rates indicate that the most remote level of the hierarchy is 6 times farther than the cache. They also predict that the potential improvement by using the prefetch unit to offset latency may bring the global memory access rate in line with the cluster memory access rate. Of course, global memory is still viewed as being farther away since there are restrictions on the addressing modes for which prefetch is possible and it must deal with 32 processor contention as opposed to the 8 processor contention within any cluster memory.

Another source of performance limitation is the control overhead within the CE due to the experimental setup. As a result of this overhead the peak effective CE data demand is reduced and the speedups that we observe can exceed the 16 given by comparing 1 GIB/CE demand rate of 5.88 MW/s to the 94 MW/s aggregate bandwidth of the 32 memory banks.

Component	Unit rate	Aggregate rate
Cluster cache	47	188
Cluster memory	23.5	94
Network link	5.9	94
Memory bank	2.94	94
GIB/CE		
w/o prefetch	$\approx$1	$\approx$ 32
w/ prefetch	5.88	188

Table 2: Hardware limiting access rates in MW/s.

Component	Slowdown
Cluster cache	1
Cluster memory	2
Global memory	
w/ prefetch	2
w/o prefetch	$\approx$ 6

Table 3: Relative performance degradation of memory hierarchy based on hardware rates.

In our experiments the multicluster control overhead has been reduced to negligible levels due to our steady state assumptions. However, the control overhead due to the dynamic scheduling of the iterations within cluster and the looping within a CE represent overhead that is *real* in the sense that it is unavoidable any time the memory system is used in practice. We have kept this as small as possible so that any further overhead incurred in a particular practical situation may be modeled by the NOP parameter.

This overhead has been estimated in two ways. The first – a static cycle count of the code generated for the experiments – results in a prediction of 3.4 MW/s per CE for a prefetch block of size 32 and 4.7 MW/s per CE for a prefetch block of size 512 when using prefetch in global memory. (The prediction assumes a very simple assignment of iteration blocks to processors.) This yields aggregate CE-induced limits of 108 MW/s and 150 MW/s respectively. These are still over the limiting bandwidths of the global hardware but are more reasonable peaks to compare against when discussing the balance of the system. (The cycle counts above apply to the code which can use multiple CE's per cluster. When only 1 CE per cluster is used the overhead is slightly lower.)

These limits can also be tested empirically with a simple modification of the template of Figure 2.1.2. The innermost loop iteration, P4, is altered to operate on register data rather than global memory. This simulates a global memory that does not suffer any bandwidth degradation due to multiple processor accesses. Latency is modeled by issuing a single scalar read from global memory at the beginning of the loop which processes a prefetch block, i.e., after P2. Table 4 shows the empirically determined CE overhead limits to achievable bandwidths. It is these bandwidths that should be used to determine how well the global memory is doing when it is not saturated. The values show that, as expected, the cycle count predictions are somewhat optimistic. Note that for a prefetch block of size 32 the bandwidth for 32 CE's is very close to the hardware limit of global memory.

Total CE's	Clusters	pf=32	pf=512
1	1	3.3	4.6
2	2	6.5	9.2
3	3	9.8	13.7
4	4	13.0	18.2
8	1	26.0	35.0
16	2	51.0	70.0
24	3	75.4	105.0
32	4	90.1	140.0

Table 4: Basic global memory vector read overhead bandwidths in MW/s.

2.3 Results

In this section, we present some of the results of the basic performance of the memory system. We concentrate on the LOAD kernel from the various locations in the memory system.

2.3.1 Basic memory performance

The performance of the cluster memory hierarchy has been characterized in detail and an attendant performance prediction strategy developed elsewhere, [1, 2], and will not be repeated in detail here. As expected, the performance varies considerably for each of the parameters listed above. For comparison purposes, Table 5 contains the basic performance of a single cluster memory system for a stride 1 access using an iteration block of 512 (as is used for the global memory experiments) and 0 NOPS. The cache bandwidths are much more sensitive to the variation of the experimental parameters than the cluster memory bandwidths and therefore the value of 34 MW/s is somewhat optimistic in practice. Rates in the high 20's are more typical. Note that, not surprisingly, the observed performance is significantly lower than the cluster hardware limits in Table 2; a factor of 2 for the memory and between 1.4 and 2 for the cache. The relative degradation of observed performance for cluster cache to that of cluster memory, roughly 2 - 3, is about the same as the factor of dictated by the hardware limits (see Table 3). The bandwidth limitation of the cluster memory as the number of CE's increases is also apparent from the observed performance.

The most basic way to access global memory on Cedar is in vector mode without prefetch. Since the memory system is designed to handle requests issued at a much higher rate there is no real difficulty with degradation due to network and memory bank contention. Table 6 contains the performance for various cases of the basic experimental configuration, i.e., iteration block size of 512 and 0 NOPS. The speedup relative to a single processor accessing global memory without prefetch is included. Clearly, contention does

Type	Processors		
	1	4	8
Cache	4.5	18	34
Memory	2.3	9.3	12

Table 5: Basic cluster read bandwidths in MW/s.

not occur intensely enough to cause any significant deviation from linear speedup as the number of CE's increases.

The performance slowdown relative to the observed performance of the same number of processors in the same number of clusters accessing their local caches is also given in the table. Note that it is less than the degradation factor of approximately 6 implied by the hardware limits (see Table 3). This is due to the fact that without prefetch the global memory system degrades less from its hardware limit than the cluster memory system. The cluster memory system degradation factor, however, can vary significantly due to the sensitivity of the cache access to the experimental parameters and a slightly lower figure of 3.5 to 4 is probably more reasonable in practice.

Total CE's	Clusters	Bandwidth	Slowdown
1	1	.98 (1.0)	4.6
2	2	1.9 (1.9)	4.7
3	3	2.9 (2.9)	4.7
4	4	3.8 (3.9)	4.7
8	1	7.7 (7.8)	4.4
16	2	15.4 (15.7)	4.4
24	3	22.6 (23.1)	4.5
32	4	29.4 (30.0)	4.6

Table 6: Global memory vector read w/o prefetch bandwidths (speedups) and degradations relative to observed cache bandwidths.

As is indicated by the hardware limits, the best way to access global memory is to make use of the prefetch capabilities of the GIB/CE. The simplest mode of this type of access is when all of the CE's access locations starting in memory bank 0 and proceed using a stride of 1. Table 7 contains the bandwidths and speedups for the basic experimental configuration for the two extreme prefetch block sizes and an implicit prefetch.

For 1 CE there is virtually no contention and the bandwidths agree with the cycle count predictions given above. There are several important results indicated in the table. Note that for a small number of CE's the speedup is essentially linear, as one would expect, and increasing the size of the prefetch block used can improve performance significantly. As the number of processors involved increases, contention increases degradation and the influence of the size of the prefetch block diminishes rapidly. Notice also that the distribution of the CE's is important when considering the prefetch block size. When 1 CE per cluster is used the influence of the prefetch block size reduces very quickly – at 4

Total CE's	Clusters	pf=32	pf=512
1	1	3.2 (1.0)	4.4 (1.4)
2	2	6.3 (1.9)	8.8 (2.8)
3	3	9.4 (2.9)	11.0 (3.4)
4	4	12.0 (3.8)	13.0 (4.1)
8	1	21.5 (6.7)	30.9 (9.7)
16	2	37.2 (11.6)	46.2 (14.4)
24	3	48.3 (15.0)	52.1 (16.3)
32	4	54.7 (17.1)	54.4 (17.0)

Table 7: Basic global memory vector read bandwidths in MW/s (speedups).

CE's the difference between 32 and 512 is negligible. But, when 8 CE's are used within a single cluster, the difference in performance between 32 and 512 is still significant. This is due to the initial shuffle which is present in the network between the CE's and the first stage crossbars. As a result, if we assume that CE 0 is used in each cluster for the 4 CE 4 cluster data point all 4 CE's are competing for the 4 output ports of the same crossbar. Therefore, contention dominates over the effect of the prefetch block size. When 8 CE's in 1 cluster are used, the CE's are distributed in pairs across all 4 first stage crossbars. Each pair then has 4 output ports available for use resulting in significantly less contention in the first stage of the network. It is also clear that contention levels off the performance of the memory system once the number of CE's exceeds 16.

Table 8 shows the slowdown of observed global memory bandwidth relative to the observed cache bandwidth with the same CE/cluster configuration. It is clear that for a small number of CE's when the network is not a problem the global memory can approach the rate of the cluster cache. Data in the prefetch buffer on each GIB can be accessed in essentially the same amount of time as a cluster cache. So after paying the round trip latency cost for the first element of the prefetch block the rest are accessed at cache rates. (For a small number of CE's stalls due to elements returning to the prefetch buffer out of order are few.) As the number of CE's contention begins to play a role and the performance falls between that of cluster cache and cluster memory. For some points it is still significantly better than cluster memory, e.g., 8, 16 and 24 CE's have slowdowns relative to cache of 1.1, 1.5, and 1.9 respectively. As the number of CE's approach 32 the performance is only slightly better than cluster memory.

The surprisingly good performance of the global network compared to cluster memory can be attributed to two facts. First, the global network is more robust in the face of contention since it is an Ω-network as opposed to the bandwidth limited bus used to communicate between cluster memory and cluster cache. Second, transactions between the CE and the cluster memory are processed with only 2 outstanding memory requests per CE possible. The number of outstanding requests per CE in the global network is determined by the prefetch block size and can be as large as 512. The transfer between the prefetch buffer and the CE is done with 2 outstanding requests, but as noted above, the cost of a prefetch buffer access is about the same as a cluster cache access. So until contention slows down the rate at which the prefetch buffer is loaded from global memory the GIB/CE combination has clear advantages over the CE/cluster memory combination.

This clearly demonstrates that in practice the prefetch capabilities of the architecture can be used effectively to offset global memory latency. One must keep in mind however, that the speed of access to global memory is not without its constraints. The prefetch unit must have a simple linear address function to work with in vector mode, i.e., vector gathers and scalar accesses can not use the prefetch buffer address generation capabilities. In this respect the cluster memory is more flexible. Also note that the cluster memory bandwidth must scale in a perfectly linear fashion with the number of clusters whereas the global memory bandwidth increase is clearly degrading as the number of processors increases.

Total CE's	Clusters	pf=32	pf=512
1	1	1.4	1.0
2	2	1.4	1.0
3	3	1.4	1.2
4	4	1.5	1.4
8	1	1.6	1.1
16	2	1.8	1.5
24	3	2.1	1.9
32	4	2.5	2.5

Table 8: Basic global memory vector read bandwidth degradation relative to observed cache bandwidths.

2.3.2 Degradation mitigation

One of the issues that the experimental configuration attempts to address is the effect of the density of requests on the performance of the global memory system. We might, for example, want to answer the question: *What ratio of global and cluster accesses is needed in order to see near linear speedup in global memory bandwidth as the number of CE's increase?* or *For a fixed density of requests how many processor can the global memory system support before contention causes significant performance degradation?* These situations are modeled by the NOPS parameters in the templates. As the number of NOPS per iteration block increases the density of requests on the network decreases. These NOPS can be used to represent a CE performing work involving data from cluster memory or from registers.

There are several ways to look at bandwidth and speedup as the sparsity of requests and the configuration of Cedar change, each of which yield a particular piece of the puzzle. Of course, as the number of NOPS increases the bandwidth achieved must be nonincreasing since more work is performed for the same amount of data transferred from global memory. The rate at which the bandwidth decreases, however, provides insight into the balance between cluster and global memory activity required for the global memory bandwidth to scale in a reasonable fashion. Figure 3 plots the observed bandwidth against the sparsity of the access for different total number of CE's using a prefetch block size of 512. The sparsity, ρ, is NOPS/512 and measures the number of extra cycles added per data element fetch in the block of 512 elements that a CE fetches on a single iteration using various prefetch block sizes. If $\rho = 0$ then there are no added dead cycles, i.e., the access is as dense as possible. As ρ increases the sparsity of the access increases and the amount of contention is decreased. (Of course, the value of ρ only reflects the sparsity within an iteration block. The other cycles due to loop control and other CE overhead also contribute dead cycles to increase sparsity but those are fixed for all of the experiments.)

If contention is not a significant factor the observed bandwidth should be a hyperbolically decreasing function of ρ as is seen for $p = 1$ to $p = 8$. When contention is a problem, however, increasing the sparsity may reduce contention and the general shape of the bandwidth curve is decreasing but concave. This is due to the fact that the reduction in contention improves the effective bandwidth more than the additional cluster work, represented by NOPS, decreases the effective bandwidth. The more intense the contention for the configuration the longer the concave region persists along the sparsity axis. The variation in concavity is clearly seen by comparing the $p = 16, 24$, and 32 curves. The increasing number of CE's entails an increased amount of contention and a larger concave region.

Figure 4 presents another view of the effect of sparsity. It plots bandwidth against CE's for various sparsity values ρ. To answer the first question above one would look for the value of ρ beyond which the bandwidth increased nearly linearly. The more linear the curve the less degradation due to contention. Clearly, as ρ approaches 2 the curves have an essentially linear profile against the number of CE's. The second question is answered by noting the number of processors on each curve, P_{crit}, where the bandwidth begins to level off thereby degrading the desired near linear behavior. The expected trend of P_{crit} increasing as ρ increases is evident in the figure.

As the sparsity of requests increases, the importance of the behavior of global memory in determining speedup over 1 CE decreases. This is clear, for example, when the NOPS are used to represent the amount of work performed by the CE's that involves only cluster memory and registers. In the limit, for very large values of NOPS the speedup observed for these templates should be equal to the number of CE's in use, p. Figures 5 and 6 plot speedup over 1 CE as a function of ρ for a given number of CE's for prefetch block sizes of 32 and 512 respectively. If contention is not significant the curve should be horizontal at the value of p. Deviation from horizontal indicates the magnitude of the degradation due to contention. The near horizontal behavior of the curves for small p shows that there is not enough activity to cause serious contention even with very dense global memory traffic. As p increases, the degradation from horizontal becomes more extreme. Also note the difference in trends caused by the change in the prefetch block size. Prefetching 512 elements places more of a strain on the network since there is relatively less sparsity caused by CE overhead. As a result, degradation of performance occurs with fewer CE's than when prefetching 32 elements. Also, the rate at which the curve moves to horizontal for a given increase in ρ is greater when prefetching 512 elements.

The results above address the issue of balance between global and cluster memory activity. However, for a given code running a particular problem this is a fixed ratio. We next address the issue of manipulating contention in this case. We will take the worst case sparsity from above, i.e., $\rho = 0$ and attempt to improve performance of the steady state global memory bandwidth. Of course, we must still access the same locations, but we do have degrees of freedom concerning what elements each CE will fetch. This can be controlled via the offset from bank 0 of the initial element accessed by a CE and the stride used to access the remaining data elements assigned to the CE. The experiments above all assumed stride 1 and an offset of 0 for all CE's, i.e., all CE's start their accesses in bank 0.

We first consider the offset of the initial data element. For the given network, it is easy to determine an offset for processor p of cluster c, where $0 \leq p \leq 7$ and $0 \leq c \leq 3$, such that the initial conditions of the access have minimum contention. This offset is $\sigma = 8c + p$. Contention in each of the first stage crossbars of the forward network is as small as possible. The 8 input ports of each separate into pairs such that each pair uses a different output port of the 4 available. The use of σ The input/output ports of each second stage crossbar of the forward network match up in a similar optimal fashion. Each of the 4 input ports fanout to two a distinct pair of output ports. The mirror image configuration applies on the reverse network. As a result, we have the minimum possible contention on the initial conditions, a 2-to-1 fan-in followed by a 1-to-2 fan-out.

The stride parameter then determines how this initial condition contention evolves over time. Given that we have 32 CE's and memory banks, a stride of 32 would maintain this initial condition throughout the accesses and minimize contention. For a stride of 1, each CE pair accesses the same output port of the first stage crossbar for 8 consecutive elements and then moves to the next output port for 8 more accesses and so on cycling through the 4 output ports of the crossbar. The use of all 4 output ports by each of the input ports means that all 8 input ports interfere with each other causing contention very early in the network. Since each first stage output port connects to a crossbar that services 8 memory banks, the maximum number of consecutive stride 1 accesses that can use a particular port is 8. This implies that any first stage contention is maintained for a relatively long period of time.

A stride of 8 implies that the CE pairs still cycle through the 4 output ports, but they do so at a much higher frequency by changing output ports on every data element access. As a result, the amount of continuous time any particular CE needs any particular port is the smallest possible – one element access. This potentially allows contention to be more smoothly balanced through the system. Finally, a stride of 16 takes the first step toward the maintenance of the initial condition for all accesses that is achieved with a stride of 32. Each pair of input ports only accesses 2 of the output ports – alternating between them on each data element access. The output ports separate into 2 pairs each of which services a set of 4 distinct input ports. As a result, the overall contention in time is reduced as in the stride 32 case and the frequency of moving between output ports is increased as in the stride 8 case.

On first examining this situation, it seems clear that a stride of 32 would be the best choice since it minimizes contention over time. Unfortunately, there is a compensating effect that occurs to degrade performance as stride increases. The prefetch buffer does not translate virtual to physical addresses. This is done by the CE. When a prefetch unit, issuing physical addresses, crosses a page boundary it must halt and wait for the CE to provide the next physical address. After receiving this address, the unit can proceed to generate and access physical addresses independently of the CE. The larger the stride the more page boundary crossings that will occur in a given prefetch block. The prefetch unit, therefore stalls more and degrades performance.

Figure 7 plots the bandwidth against prefetch block size for different strides using σ as the offset in each CE. The effect of these competing trends are clear. By manipulating stride and offset the bandwidth can be improved to approximately 73 MW/s up from the

55 MW/s seen above for the no offset stride 1 accesses. Note the superiority of the strides of 8 and 16 which strike a reasonable compromise between contention and page boundary crossing. The stride 1 access does not improve performance by altering the offset from 0 to σ. Similar experiments with random offsets have shown a similar lack of sensitivity – or more precisely, they have shown that p and the prefetch block size are the important parameters for a 0 NOP stride 1 access.

There is one more way of enhancing the performance of the global memory accesses which improves considerably the performance of the 0 offset, stride 1, $p = 32$ case above. Recall that the prefetch buffer into which elements returning from global memory are placed on the GIB is a simple direct map cache with some additional restrictions related to page boundaries. As a result, if we have a spatial locality in global memory references the exploitation of this cache may aid performance. Table 9 compares the bandwidths observed for the 0 offset, stride 1, $p = 32$ access of global memory with and without a prefetch buffer hit on the data. This is accomplished by adding another vector loop within each CE's iteration. Immediately after accessing an entire prefetch block the CE issues again the same global memory addresses. These elements should reside in the prefetch buffer and are, therefore, accessible at almost cluster cache rates. The results show the expected trend of a significant reduction in the cost of the second access. As the prefetch block size increases the benefit increases since the fraction of time spent in global memory accesses rather than in control overhead is more significant.

prefetch size	read	read + hit
32	54.7	87.0
64	53.5	91.5
128	53.9	95.2
256	54.2	96.5
512	54.4	97.4

Table 9: Bandwidths for a vector read and a vector read followed by a hit in the prefetch buffer, 0 NOPS.

3 A global memory simulator

In this section, performance estimates obtained through simulation are used to determine the scalability of systems based on the Cedar shared-memory network. The term *scalable* is used here to describe systems whose per-processor performance is roughly constant across a range of system sizes. Performance is evaluated in terms of aggregate bandwidth, latency of a vector fetch request and pipeline rate of packets within such a burst request. In our experiments, systems consisting of 32 to 512 processors connected to an equal number of memories are examined.

3.1 Simulation model and calibration

Realistic models of both the target architecture and associated traffic patterns are essential. Accordingly, a register transfer level simulation model based on the existing hard-

Table 10: Network configuration versus system size

Number of: Procs.	Mems.	Switch Sizes for: Forward Network	Reverse Network
32 Cedar	32	8x4, 4x8	8x4, 4x8
32	32	8x8,4x4	8x8,4x4
64	64	8x8,8x8	8x8,8x8
128	128	8x8,8x8,2x2	8x8,8x8,2x2
256	256	8x8,8x8,4x4	8x8,8x8,4x4
512	512	8x8,8x8,8x8	8x8,8x8,8x8

ware was used, and it was driven with traffic patterns which reflect the behavior of the LOAD/STORE kernels. This simulator has been described before, and results for a more general traffic model presented, in [3].

The simulator models the data path of Cedar's global memory system in detail. This includes the GIB's, switches in both networks, and memories modules. The CE's and cluster memory systems are not modeled in detail, instead GIB prefetch block requests are used to generate the traffic patterns which load the network. Due to interactions between the CE's and GIB's which are not modeled explicitly, bandwidths observed by experimentation deviate slightly from those seen by the simulator.

The simulation results presented are based on the vector block prefetch: all CE's participate, LOAD requests are considered, and a block prefetch length of 32 words is used. The timing of the GIB prefetch block requests is determined by analysis of the LOAD STORE kernels. The delay between the completion of one prefetch request and the initiation of the next is determined by the loop overhead in the kernels, and the number of sparsity NOPS. The spatial distributions have no offset and a stride of one.

Several different traffic distributions have been run through the simulator which do not correspond directly to any generated by a load/store kernel. Many of these alternatives have no significant impact on overall performance, and some have very chaotic effects. Of particular interest, though, is the dramatic improvement in performance which can be seen by matching the GIB issue rate to the memory module service rate. This effect is difficult to duplicate in actual experiments, but yields performance improvements of up to 50% in simulations. Data for these traffic patterns will be presented in a future technical report.

The simulator was calibrated by comparing the bandwidth measured in several of the experiments presented previously to that obtained by simulation of the same traffic patterns on an identically configured system. For the cases examined the simulated results match the measured values with less than 10% deviation. For example, with all CE's participating in a vector block fetch at a sparsity of zero, Cedar achieves a bandwidth of 51 MW/s, while the simulated result is 47 MW/s.

The configurations of the networks used, in terms of the size of the switches at each stage, are given in Table 10. Cedar has so far implemented only part of the full network connectivity, but in the interests of generality, the full network is explored in detail here. Where applicable, performance data for both configurations of the 32 processor system

Table 11: Performance results for sparsity= 0

System Size	Bandwidth (MW/s)	Block Latency	Interarrival Time
Cedar	47	25.2	5.4
32	78	15.5	2.8
64	107	18.3	4.0
128	178	24.1	4.8
256	305	32.4	5.4
512	527	31.7	6.7

are given.

3.2 Metrics

Within the simulations three metrics are used. As before, aggregate bandwidth and speedups are used to evaluate overall performance. In addition, block latency and message interarrival time are used to describe the behavior of individual block fetches. These latter metrics are determined for each individual message and the arithmetic mean over all elements is then calculated.

Block latency is defined as the number of clock cycles that elapses between the time a processor begins issuing a block fetch and the time that the first message in the block is received by the processor. Interarrival time is the delay between the return of successive packets in a block to the processor that issued them.

3.3 Results

The performance results for various systems performing a vector block prefetch with a sparsity of zero are shown in Table 11. In theory, the 32 to 16 fan-in, and 16 to 32 fan-out should not limit the performance of the network, because the memories can only service requests at half the rate at which GIB's generate them. In practice, this can be seen to be untrue. By doubling the number of connections in the middle of the forward and backward networks, achieved bandwidth increases by over 50%. Only the "full" network case will be considered further.

We now consider traffic with more sparsity. In Figure 8, curves are shown which correspond to those shown previously for the Cedar system. Because of the large range of values shown, the y-axis is scaled logarithmically. All the systems achieve a good fraction of the ideal speedup for some value of sparsity, but the larger systems require larger amounts of sparsity (which is in a sense equivalent to locality of reference within a Cedar cluster) in order to offset their increased contention.

In Figures 9 and 10 a similar trend can be seen in the interarrival time and latency values. Here the x-axis shows the size of the system on a logarithmic scale to emphasize the fact that for low sparsity performance degrades as the log of system size. As sparsity (S) increases, the system scales better: for very moderate sparsity (2+), per-processor performance is almost constant across system sizes. The jump in latency seen as the size increases from 64 to 128 processors is due to the extra network stage needed to construct the larger systems (see Table 10).

It is important to remember that the number of buffers used in the Cedar network is quite low, only 3 or 4 words of buffering per stage. In previous experiments (see [3]) moderate buffering has been shown to allow much more intense traffic to perform as well on large systems as on the 32 processor Cedar machine. By increasing buffering to 16 words per stage performance scales well for high loads, and the limiting factor becomes the memory access time, which is four 85ns network cycles for the Cedar system.

References

[1] K. GALLIVAN, D. GANNON, W. JALBY, A. MALONY, AND H. WIJSHOFF, *Behavioral characterization of multiprocessor memory systems*, in Proc. 1989 ACM SIGMETRICS Conf. on Measuring and Modeling Computer Systems, New York, 1989, ACM Press, pp. 79–89.

[2] K. GALLIVAN, W. JALBY, A. MALONY, AND H. WIJSHOFF, *Performance prediction of loop constructs on multiprocessor hierarchical memory systems*, in Proc. 1989 Intl. Conf. Supercomputing, New York, 1989, ACM Press, pp. 433–442.

[3] E. D. GRANSTON, S. W. TURNER, AND A. V. VEIDENBAUM, *Design and analysis of a scalable shared-memory system with support for burst traffic*, in Proceedings of the 2nd Annual Workshop on Shared-memory Multiprocessors, ISCA-90, Kluwer and Assocs., 1991.

[4] J. KONICEK, T. TILTON, ET AL., *The organization of the cedar system*, in Proc. 1991 International Conference on Parallel Processing, Penn State UNiversity Press, 1991.

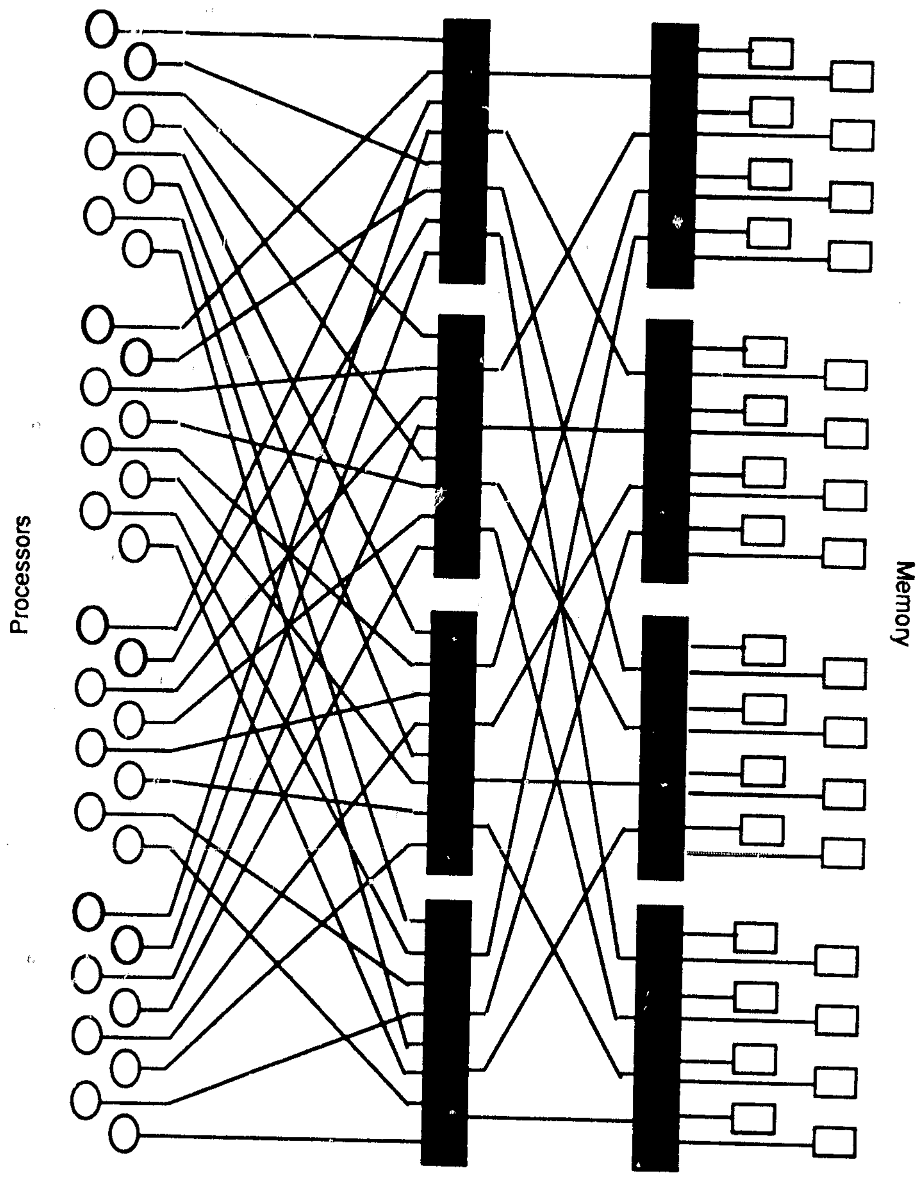

Figure 2: The Cedar interconnection network

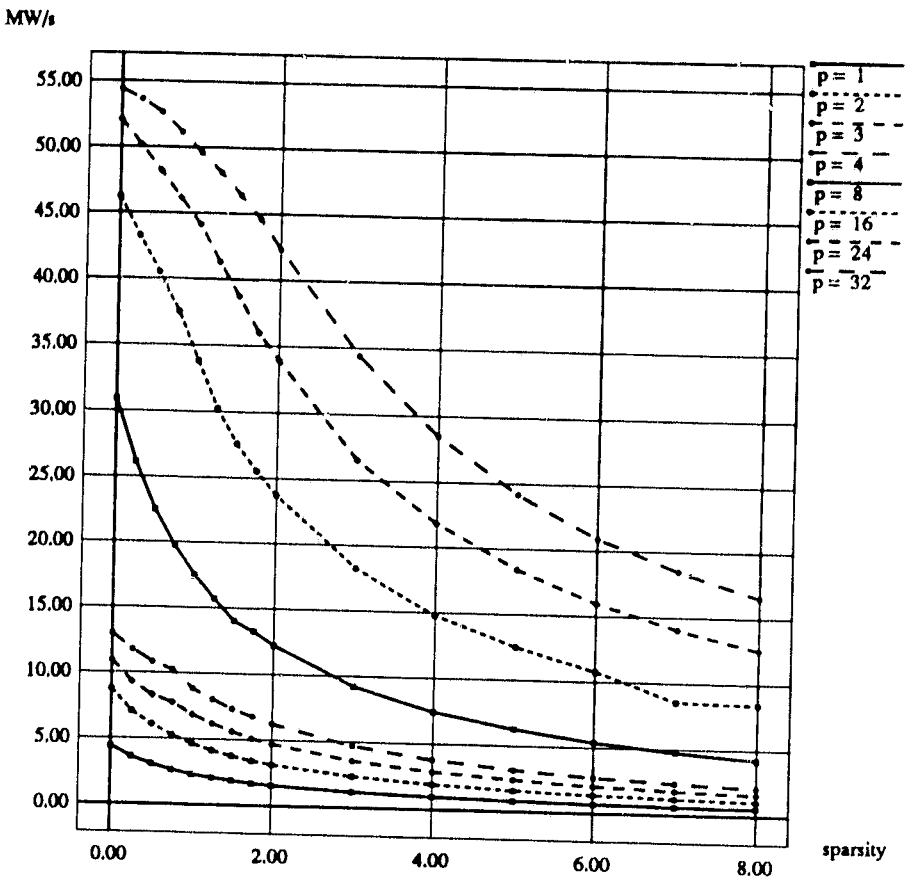

Figure 3: Implicit prefetch vector read bandwidth vs. sparsity, $pf = 512$.

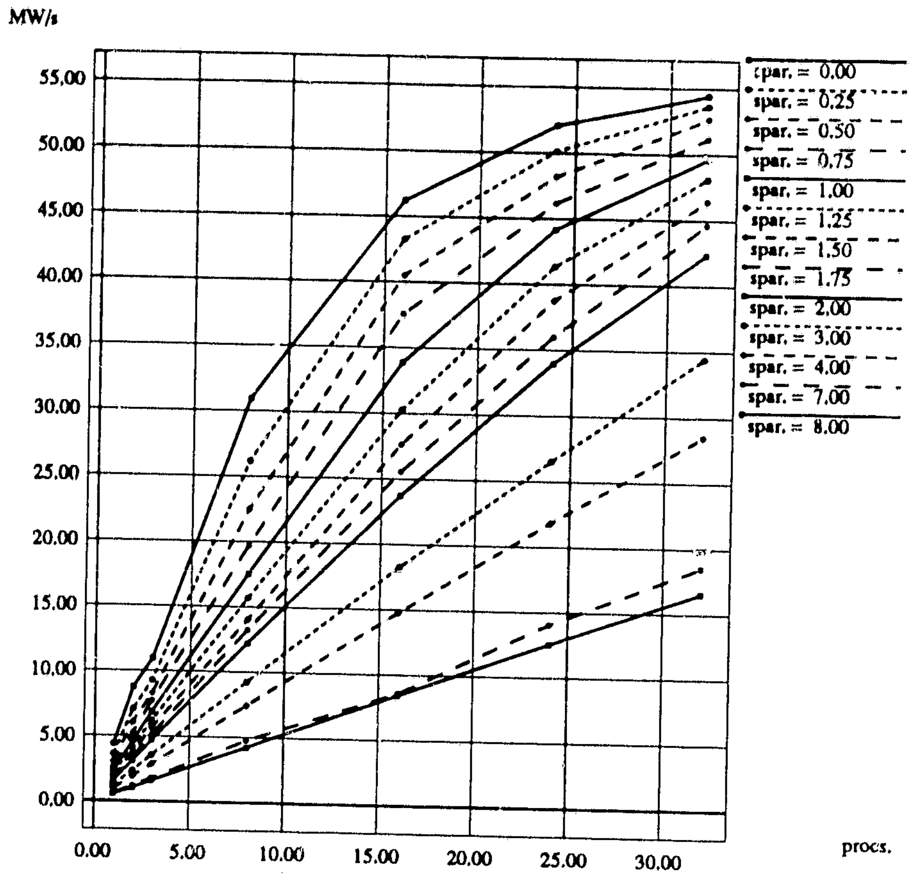

Figure 4: Implicit prefetch vector read bandwidth vs. CE's, $pf = 512$.

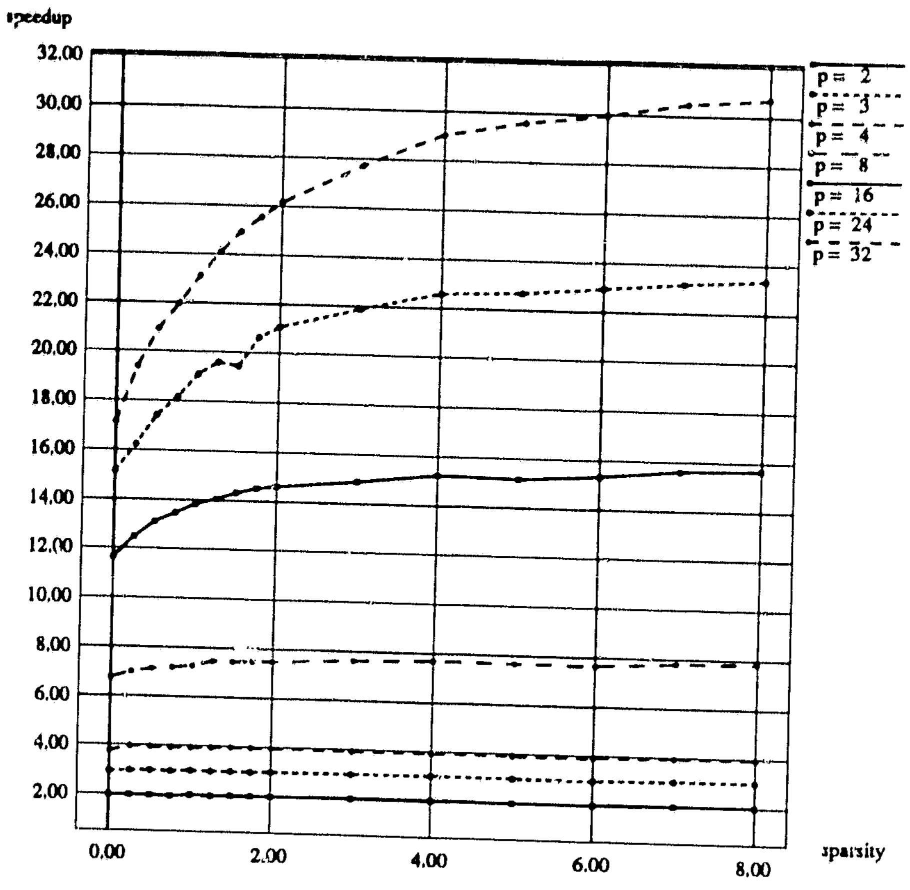

Figure 5: Speedup vs. sparsity, $pf = 32$.

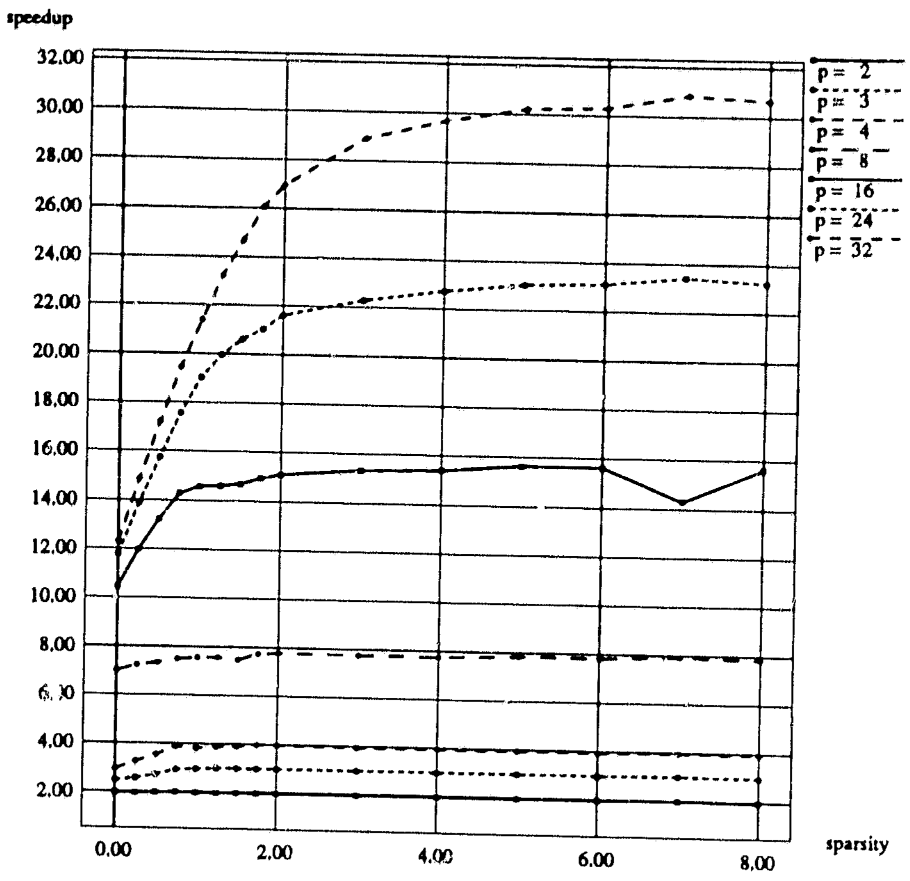

Figure 6: Speedup vs. sparsity, $pf = 512$.

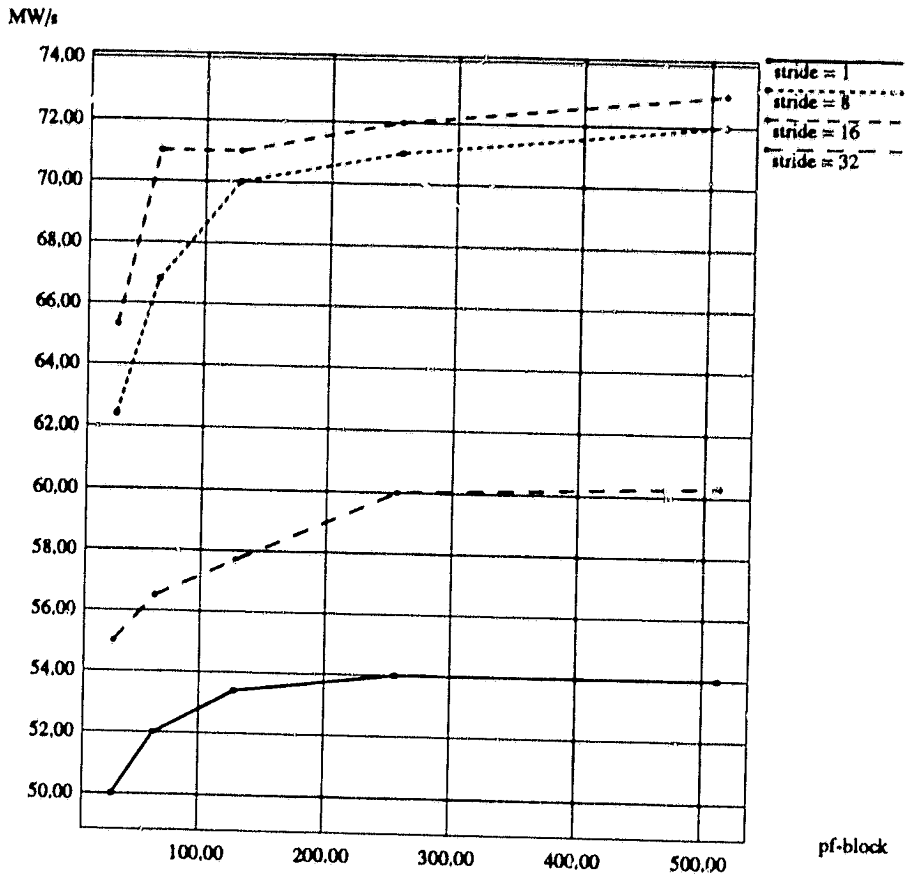

Figure 7: Bandwidth vs. prefetch size, 0 NOPS $p = 32$.

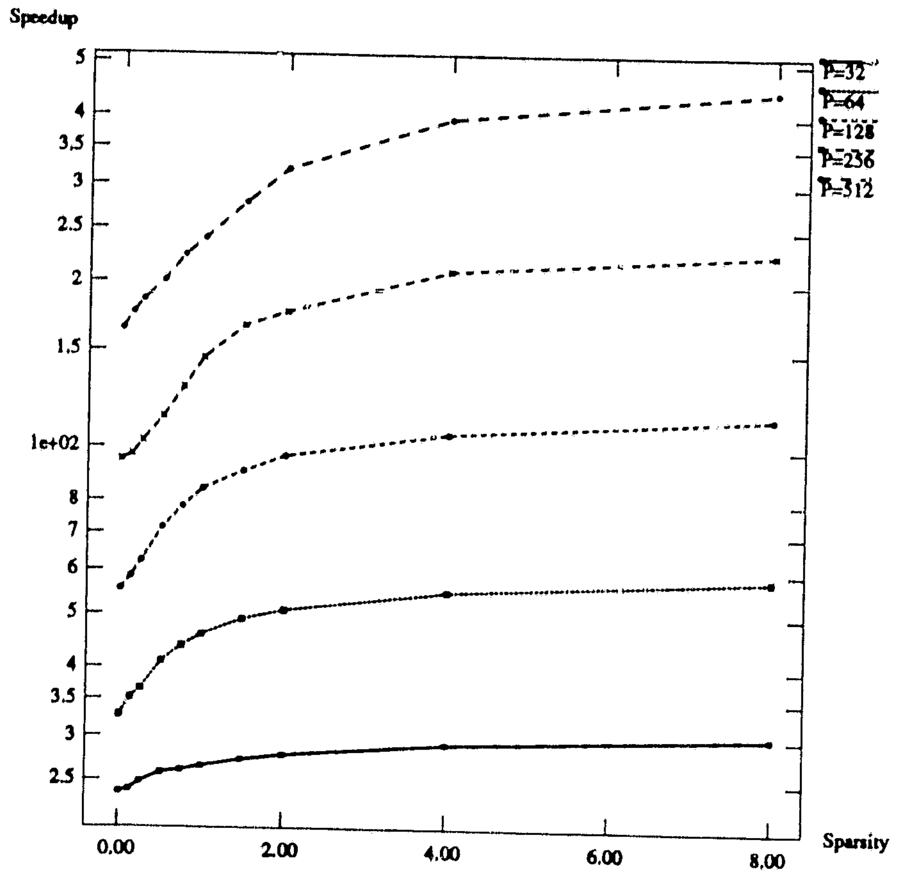

Figure 8: Speedup vs. Sparsity

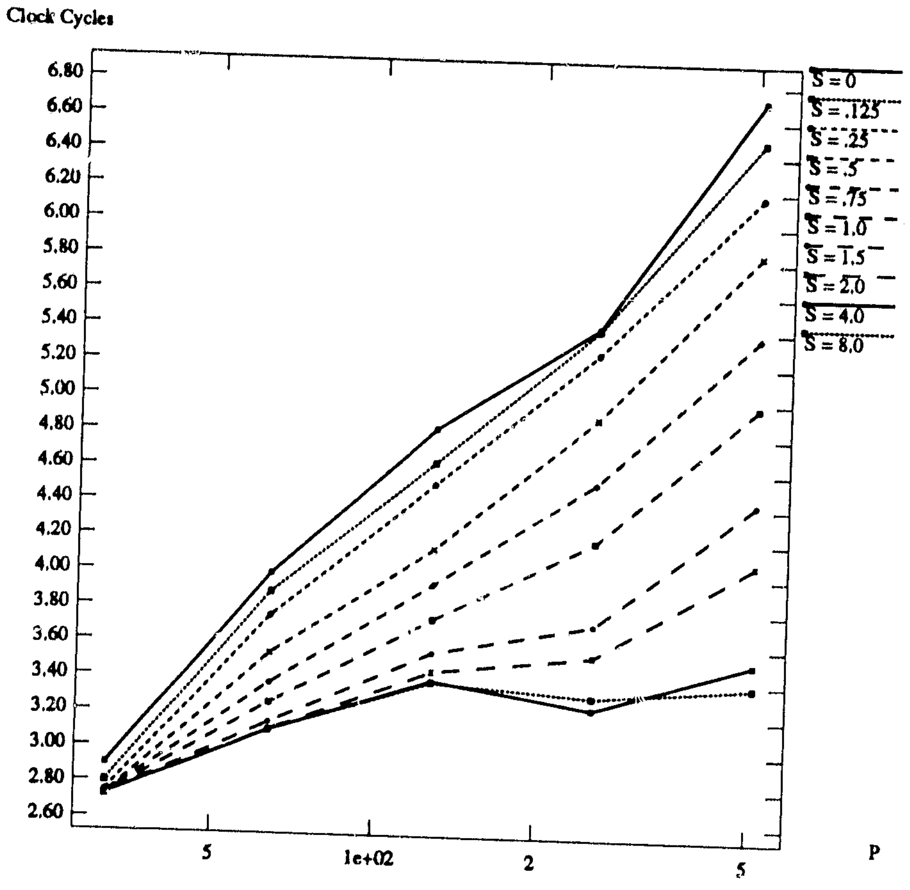

Figure 9: Interarrival Time vs. System Size

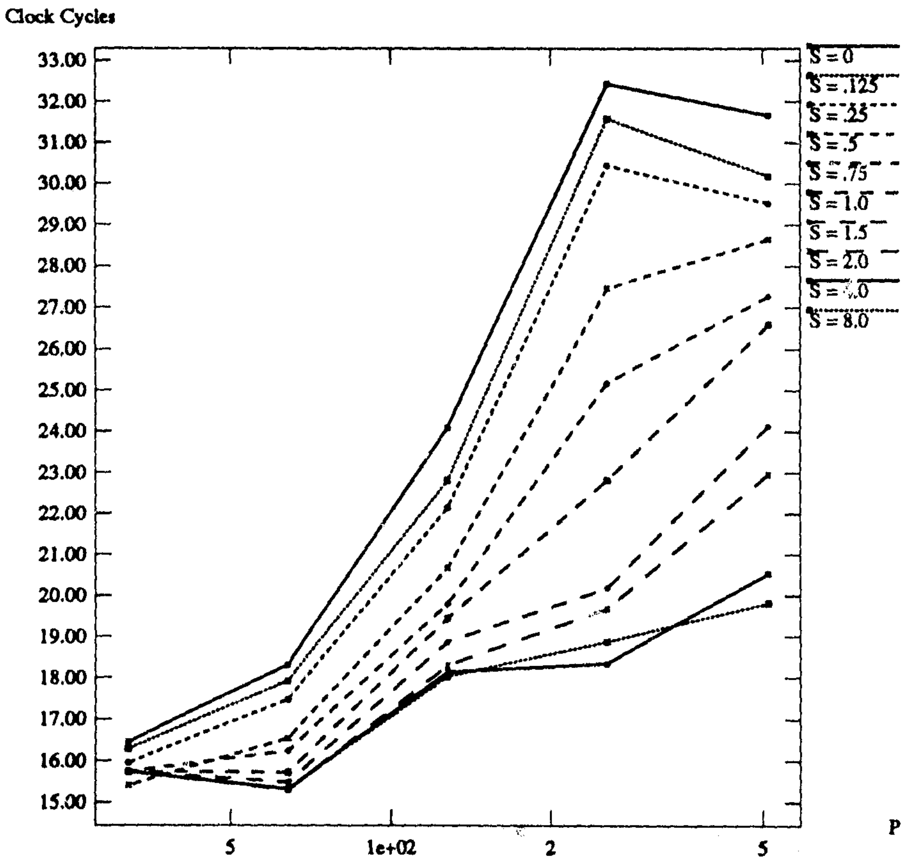

Figure 10: Block Latency vs. System Size