

# The Performance Value of Shared Network Caches in Clustered Multiprocessor Workstations

*John K. Bennett*

*Katherine E. Fletcher*

*W. Evan Speight*

March 11, 1996

Computer Systems Laboratory  
Department of Electrical and Computer Engineering  
Rice University  
P.O. Box 1892  
Houston, Texas 77251-1892

(713) 527-4020

FAX (713) 524-5237

email: {jkb, kef, espeight}@rice.edu

## Abstract

This paper evaluates the benefit of adding a shared cache to the network interface as a means of improving the performance of networked workstations configured as a distributed shared memory multiprocessor. A cache on the network interface, shared by all processors on each cluster, offers the potential benefits of retaining evicted processor cache lines, providing implicit prefetching when network cache lines are longer than processor cache lines, and increasing intra-cluster sharing. Using simulation, the performance of eight parallel scientific applications was evaluated. In each case, we examined in detail the means by which processor cache misses were satisfied.

Our results were mixed. For the applications studied, we found that the network cache offers substantial performance benefit when processor caches are too small to hold the application's primary working set, or when network contention limits application performance. The expected benefits of implicit prefetching and increased intra-cluster sharing did not contribute significantly to the performance enhancement of the network cache for most applications. Finally, the advantage afforded by the network cache diminishes as processor cache size increases and network contention decreases.

# 1 Introduction

Parallel computing using networks of workstations has increasingly become a viable alternative to large, tightly-coupled parallel machines. Commodity workstation multiprocessors are now available from a variety of vendors including DEC, IBM, Silicon Graphics, and Sun Microsystems. Moreover, these workstation vendors are adopting open interface standards, enabling the design of flexible network interfaces that can be used to connect multiprocessor workstations to create *clustered multiprocessors*. This paper evaluates the performance value of adding a cache for remote data on the network interface to augment the available workstation per-processor caches.

Adding a cache to the network interface offers several potential benefits. The primary function of the cache is to reduce the number of high latency remote references by providing more on-cluster cache than that provided by the workstation per-processor caches alone. Potential benefits specific to a shared cache on the network interface include facilitating intra-cluster sharing and providing implicit prefetching. Used in multiprocessor workstations, such a cache may increase intra-cluster sharing beyond that of direct transfers between processor caches by allowing processors on the same cluster to refill lines evicted from processor caches by conflict or capacity misses. In addition, a cache line size longer than the processor cache line size may provide implicit prefetching without polluting the processor caches. In this paper we investigate these potential performance benefits and provide a detailed analysis of how processor cache misses are satisfied for various clustered multiprocessor configurations. Using simulation, we evaluate the performance of these configurations running eight numerical parallel benchmarks: Fast Fourier Transform (one and three dimensional), Conjugate Gradient, Cholesky, LU Dense Matrix Decomposition, Radix Integer Sort, Volume Rendering, and Water. We compare the performance of a clustered architecture with caches on the network interface to a similar architecture without the additional caches. We also vary several architectural parameters in order to understand the behavior of caches on the network interface.

Our principal result is that a network cache offers performance benefit for architectural configurations where the amount of processor cache is insufficient to contain the application's primary working set, or where contention for the network creates a bottleneck resulting in very high remote reference latencies. For clusters of more than a single processor, we expected the network cache to facilitate increased data sharing. Direct processor-to-processor cache transfers, however, capture most of the available sharing. In these applications sharing of data occurs primarily over a narrow window in time, allowing the processor caches to exchange shared data before evicting it. Thus, the potential for increased data sharing in the network interface cache is limited. Even though most applications display a significant amount of implicit prefetching with network cache lines longer than processor cache lines, few applications exhibit execution time reductions as a result. However, the reason for this is not a rise in false sharing rates, even with very long cache lines. Instead, the effect of the increased transport time of longer cache lines on miss latency most significantly affects performance. The presence of large processor caches or a low contention network reduce the benefit of the network cache, but most of the eight applications studied still show some benefit over a similar architecture without caches located on the network interface. However, a network cache offers limited performance benefit when used in conjunction with both large processor caches and low network contention.

## 2 Methodology

### 2.1 Architecture

The simulated system architecture consists of several clusters of SPARC processors connected by a high speed interconnection network. Each cluster contains a tightly-coupled, single-bus multiprocessor with one to four processors per node, private per-processor caches with 4-way set associativity and 128 byte cache lines. This cluster configuration models commodity multiprocessor workstations. To connect the clusters, a dedicated network interface is added. The network interface consists of a portion of global memory containing data for which the cluster is the “home node” and dual address mappers to reduce contention between the cluster bus and the network. The interface may also contain a large cache, *the network cache*, used to maintain local copies of remote data. To allow the network cache to manage all remote coherence, inclusion is maintained between the network cache and the per-processor caches.

Because of its large size (up to 32 MBytes), the network cache would likely be constructed from standard dynamic RAM. In this study, we assume a 60ns row address access time for both the global memory and network cache, as well as the ability to do “page-mode” transfers (multiple CAS with a single RAS). Cache blocks in the network cache are multiples of the processor cache line length. Each processor cache line length portion of the network cache line is referred to as a *subblock*. The network cache is 4-way set associative. Table ?? provides the relevant system parameters used in this study. As discussed in Section 2.4, the cache sizes shown in Table ?? were adjusted for simulation purposes.

Sequential consistency is maintained both within and between clusters. Within a cluster, consistency is maintained at 128 byte subblock granularity using a simple bus-based invalidation coherence protocol similar to the Illinois cache coherence protocol [Papamarcos and Patel, 1984]. Direct transfers of data between processor caches located on the same cluster are supported. Between clusters, consistency is maintained at network cache line granularity using the same invalidation protocol. The network topology modeled is a ring (or multiple rings for the low-contention network) that supports broadcast and snooping, allowing the network interface to monitor bus and network traffic and generate appropriate coherence actions.

### 2.2 Experiments

To investigate the performance impact of network caches, we compare the performance of cluster architectures containing caches on the network interfaces (NC) to architectures without network caches (no-NC). The network interface in both the NC and no-NC architectures snoops the interconnect, but there is no data storage on the network interface in the no-NC implementation. Thus, the basic timings of both implementations are identical. In both models, a buffer per processor allows one outstanding remote access request to be pending per processor on the cluster. The size of these buffers is 128 bytes per processor for the no-NC architecture, and matches the network cache line size for the NC architecture. In both architectures, direct transfers of cache lines between processor caches located on the same cluster are supported.

We varied the number of processors per cluster from one to four in both implementations, while maintaining the total number of processors constant at 32. We evaluated network cache line sizes from 128 bytes

to 1024 bytes. Varying the network cache line size is reflected in the network latency by increasing the transfer time by 2 nanoseconds per additional byte. The network transfer unit is 128 bytes for the no-NC architecture, matching the processor cache line size.

For each simulation we report execution times normalized to the execution time for the no-NC architecture with single-processor clusters. In addition to reporting execution times, which are highly sensitive to the particular architectural parameters chosen, we compare how processor cache misses are satisfied. Processor cache misses can be satisfied by another processor cache (in cluster configurations with more than one processor), from the cluster memory, from the network cache, or remotely from another cluster. Memory and network cache accesses are classified as hits or misses, and accesses to the network cache are further categorized as defined in Table ?? . A more detailed description of these categories can be found in [Bennett et al., 1995a].

We examine the performance impact of the network cache when two emerging technological trends are introduced: workstations with large, per-processor second-level caches; and low-contention networks. We begin by analyzing the performance impact of the network cache in a system built from current technology. This system incorporates 64K processor caches and a high-contention, 4-Gigabit network. The contention for this network results in an average network queue length of 14.2 across all applications. Next, we model a low-contention network (average queue length zero) by providing a separate 4-Gigabit channel for each processor in the system. Thus the low contention network has the same latency as the high contention network, but alleviates bottlenecks caused by contention for a single channel network. Recent technological and engineering advances have made hooking parallel channels together feasible in the near future [Xiao and Bennett, 1995]. We then examine how the impact of the network cache on system performance changes with large per-processor, second-level caches (2 Mbytes) and high network contention. Finally, we show the effects of both low network contention and large second-level per-processor caches on the performance of the network cache systems.

## 2.3 Applications

The Cholesky, Water, Radix, LU, and Volrend programs are from the SPLASH-2 benchmark suite [Woo et al., 1995]. CG and FFT-3D are from the NAS benchmarks [Bailey et al., 1991], and FFT-1D was developed locally using the algorithm described in [Press et al., 1988]. Table ?? gives shared data sizes and input sets used for each of the eight applications.

All the C applications are compiled with “gcc -O2” version 2.5.8. The one FORTRAN application, CG, is compiled with “f77 -O”, version 1.4, patch release 3. Process creation and synchronization is expressed using the ANL macros [Boyle et al., 1987].

## 2.4 Simulation Environment

All actions of the caches, busses, and network are simulated using an execution driven simulator derived from RPPT [Dwarkadas et al., 1994] on SPARC processors. Only shared data is modeled; instructions, stack, and other private data are assumed to have a 100% hit rate. Synchronization variables are placed in a separate

synchronization cache and do not conflict with ordinary data accesses. Contention and associated delays for buses and the network are accurately modeled.

To achieve a balanced percentage of local to remote accesses across all clusters in the system, memory is evenly allocated, round-robin, by network cache line among the clusters. The first network cache line referenced by any processor is allocated to Cluster 0, the second network cache line referenced is allocated to Cluster 1, etc. This method of memory distribution produces a balanced data distribution across the clusters of the system, but does so at the expense of limiting any spatial locality between network cache lines.

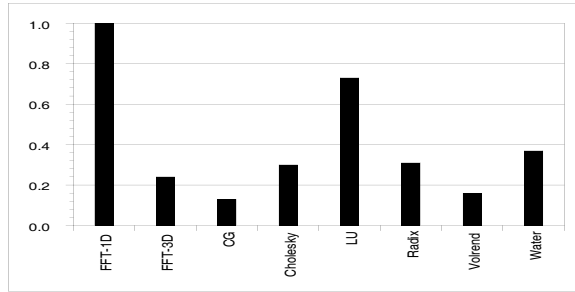
**Simulated Cache Sizes** Because our experiments are performed using simulation, we are limited to the small application sizes shown in Table ???. To model cache behavior representing the much larger problem size that would realistically run on a 32 processor machine, we scale down the processor and network cache sizes with the application problem size simulated. Recent work investigating cache working sets, and how they grow with problem size, guide these cache size determinations [Rothberg et al., 1993, Weber, 1993, Speight et al., 1994]. For each application, we determined which working set would likely fit in a 64 Kbyte primary cache and a 2 MByte secondary cache for the projected “real” problem size [Bennett et al., 1995b]. Thus, we simulate cache sizes that accommodate the same logical working sets for the simulated problem sizes. This method leads to application-specific choices of cache sizes (given in Table ??) for the simulated problems. The simulated cache sizes modeling a 64 KByte primary cache with no secondary cache are shown in column 1 (labeled “small”) of Table ???. Column 4 (labeled “large”) gives the simulated cache sizes used when 2 Mbyte second level caches are added to each processor. To simplify simulation of the architecture with secondary caches, we simulated a combined primary and second level cache with a single cycle access time. Although maintaining a one cycle hit time to such a large cache is unrealistic, our purpose is to compare the performance with that of a much larger and slower network cache and the simplification respects the relative difference in access times.

In addition to scaling the caches down for small problem sizes, we use the working set analysis to scale up the network cache size for clusters of more than one processor, ensuring that the network cache holds the same working set that would fit in the projected network cache size. For the eight applications studied here, the real network cache size of 32 MBytes accommodates the same working set for single-processor clusters as for clusters of two and four processors.

## 3 Results

### 3.1 Small Processor Caches with a High Contention Network

In this section we examine the performance benefits of a shared network cache when used with a system modeling small (64 Kbyte) processor caches and a single-channel 4-Gigabit network. We present ratios of the execution time with the network cache implementation (NC) to that without network caches (no-NC) for each of the eight applications studied. We then categorize the accesses to the network cache to gain insight into the benefits it affords.

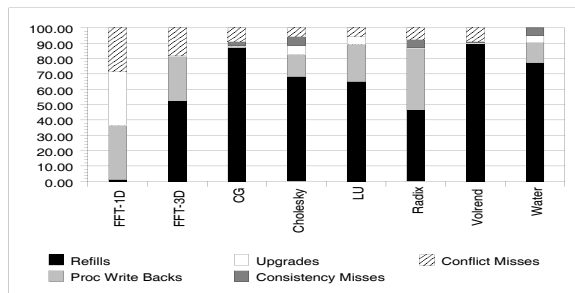


**Figure 1 Execution Time Ratio (NC/no-NC)**

Small processor caches, high network contention, one processor per cluster, 128 byte lines

Figure 1 shows the execution time with the NC implementation relative to that of the no-NC case for one processor per cluster and a network cache line size of 128 bytes. Most applications exhibited significant improvement with the addition of network caches. The execution time for six applications (FFT-3D, CG, Cholesky, Radix, Volrend, and Water) was less than 40% of the no-NC configuration. LU moderately improved, and FFT-1D did not benefit from the network cache.

Figure 2 depicts a categorization of network cache accesses using the taxonomy of Table ?? . In the sections below, we use Figure 2 to discuss the three main potential benefits of the network cache: refilling of evicted processor cache lines, implicit prefetching, and increased intra-cluster sharing. The sum of the categories comprise all accesses to the network cache .The network cache hit and miss rates can be read directly from Figure 2 because hits are displayed together at the bottom of the chart, and misses begin with the **consistency misses** category.



**Figure 2 Network Cache Data Accesses**

Small processor caches, high network contention, one processor per cluster, 128 byte lines

### 3.1.1 Refilling of Evicted Processor Cache Lines by Network Caches

Refills are hits to network cache lines by processors that have accessed the same subblock since the network cache line was retrieved. Processors access the same subblock for two reasons: processor cache capacity or conflict misses, and processor cache write backs of dirty subblocks.

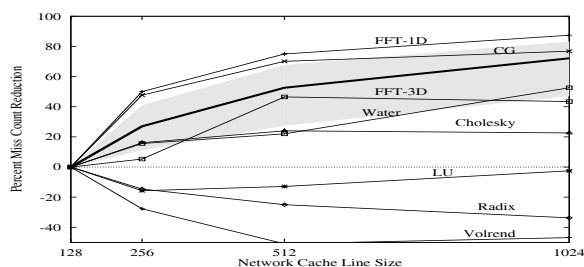
Seven of the eight applications had a significant percentage of network cache accesses that fell into the

**refills** category, as indicated by Figure 2. Processor cache capacity or conflict misses can be refilled by the network cache when it contains a larger working set than that in the processor caches, as happens for five of the eight applications (FFT-3D, CG, LU, Volrend, and Water). As expected, the execution time of these applications decreased in the configurations with network caches. Although Radix and Cholesky do not have well-defined working sets, a network cache is able to hold some additional data long enough to be reused by the processors and thus decrease execution time. Finally, network caches are unable to hold any additional data in FFT-1D due to the size of the working sets involved. FFT-1D therefore benefited the least from network caches.

Processor cache write backs may be merged in the network cache, reducing the number of write backs over the network and decreasing write latency and network traffic. The **proc write backs** category in Figure 2 shows write backs from the processor caches to non-local data. Figure 2 suggests that network caches reduced the number of long-latency write backs for six out of the eight applications. For Radix, the reduction in the number of write backs to non-local data accounts for most of the benefit provided by network caches. In the other applications, the refills of evicted processor cache lines played a more important role than write back reduction in improving execution time.

### 3.1.2 Network Cache Line Size

In this section we examine the effect of network cache line size. Longer network cache lines offer several potential performance benefits, including an increase in implicit prefetching rates, and a reduction in the number of upgrades. These two benefits must reduce the remote miss rate enough to overcome the potential disadvantages of longer network cache lines: an increase in false sharing, more contention for access to the network, and a larger network transport time. For the majority of the applications studied, longer network cache lines did not improve execution time.

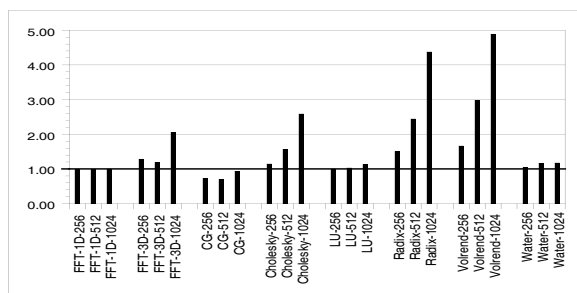


**Figure 3 Miss Count Reduction with Longer Network Cache Lines**

Two costs are associated with each network transfer: a fixed cost due to network packet headers and trailers, packet formulation, and processing delays at the receiving end; and a per-byte transport cost. Figure 3 depicts the reduction in remote misses necessary to overcome the cost associated with using a larger network cache line, in the absence of contention. The solid black line defines the minimum percentage of misses that must be eliminated for a system with a longer line to perform comparably to a system with a line size of 128 bytes. As an example, Figure 3 suggests that for an application to achieve the same performance

with 512 byte network cache lines as 128 byte network cache lines, there must be 49% fewer misses in the network cache. The values used to derive this “break-even” point come directly from our simulated system parameters listed in Table ???. The gray shaded region forms the “envelope” of possible break-even curves obtained by reducing either the transport time or the fixed time by up to three-quarters of the values given in Table ???. The actual miss count reductions for the one processor-per-cluster case for all applications are also displayed in Figure 3.

Figure 3 leads us to conclude that given current network parameters, only FFT-1D and CG exhibit enough reduction in their remote miss counts to warrant longer lines in the network cache. All other applications fall below the break-even line, and are therefore not expected to perform better with longer network cache lines.



**Figure 4 Execution Time Ratio of Network Cache Lines Relative to 128 Byte Lines**  
Small processor caches, high network contention, one processor per cluster

Figure 4 shows how the execution time of each program actually changed when longer lines were used in the network cache. For each application and line size, Figure 4 depicts the ratio of execution time of the longer network cache line architecture to that of the 128 byte cache line implementation. Thus, bar heights below one indicate that the application performed better with longer lines. Seven of the eight applications studied did not perform better with longer network cache lines. Figures 5 and 6 show the network cache data accesses for all eight applications with longer network cache lines, and indicate that while longer network cache lines did implicitly prefetch data, there was usually a corresponding decrease in the number of refills. This caused the increased transport times associated with long network cache lines to degrade performance.

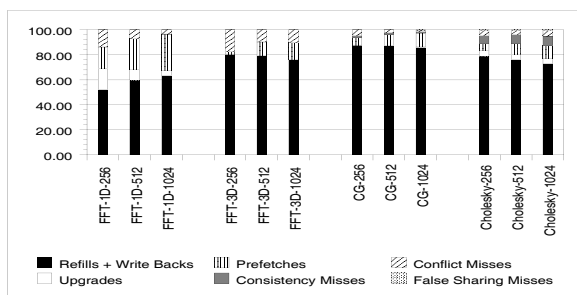
CG improved slightly with longer network cache lines. We see in Figure 5 that the number of prefetch hits in the network cache rose between a line size of 256 and 512 bytes, without a significant reduction in the number of refill hits. This lowered CG’s remote miss rate, and correspondingly reduced execution time.

Figure 3 indicates that FFT-1D has enough reduction in miss rate to make effective use of longer network cache lines, however, we see no reduction in execution time in Figure 4. This is because the added contention for the network introduced by longer lines forced nodes to wait longer periods of time for the network to become available.

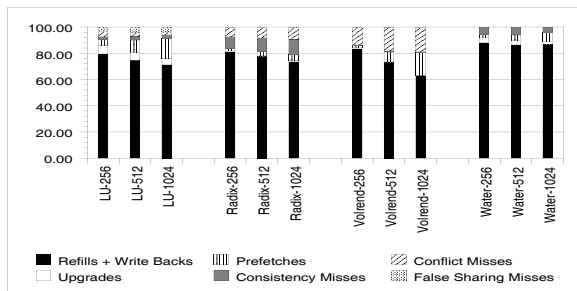
Radix, Volrend, FFT-3D, Cholesky, Water, and LU performed worse with longer network cache lines, as predicted by Figure 3. The network cache miss rate increased for Radix and Volrend (see Figure 6), due mainly to a reduction in the percentage of processor evictions and write backs that were refilled by the network cache. FFT-3D performed better with 512 byte lines than with 256 byte lines. However, this

improvement did not bring the execution time below that of the 128 byte network cache line, despite the increase in implicit prefetches evident in Figure 5. Cholesky and Water had a slight reduction in miss rate due to a rise in the amount of data implicitly prefetched with longer network cache lines, but the miss rate reduction was not enough to overcome the larger transport time associated with longer lines.

LU was the only application to demonstrate a significant amount of false sharing in the network cache. False sharing rates are calculated on a subblock basis, and therefore false sharing within a 128 byte subblock is not included in the false sharing category. The consistency miss category captures these intra-subblock false sharing misses. Taking both the false sharing and consistency categories into account, we note that although consistency misses are most of the misses in four of the applications (Cholesky, LU, Radix, and Water), this portion of the accesses to the network cache did not increase appreciably with longer network cache lines. This indicates that the longer transport time is the main reason for the poor performance of longer lines in the network cache.



**Figure 5 Network Cache Data Accesses with Larger Network Cache Lines**  
Small processor caches, high network contention, one processor per cluster.



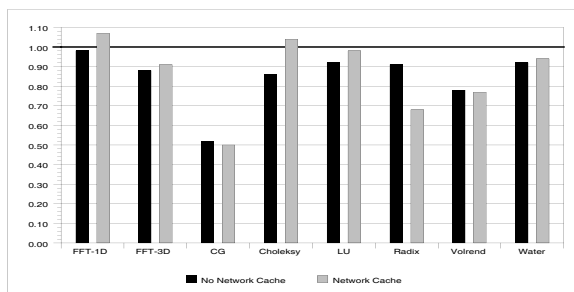
**Figure 6 Network Cache Data Accesses with Larger Network Cache Lines**  
Small processor caches, high network contention, one processor per cluster.

Although FFT-1D does not exhibit improved execution time with longer network cache lines, the data accesses shown in Figure 5 indicate another possible benefit of longer lines in the network cache: the reduction of upgrades. Upgrade hits are requests for exclusive access to a network cache line already in the cache. Therefore, although no data is transferred, a network transaction to invalidate remote copies is still necessary. Longer network cache lines in the network cache gain exclusive access to larger regions and thus reduce the number of upgrade hits. This reduction can be clearly seen in Figure 5 for FFT-1D. This effect is not present

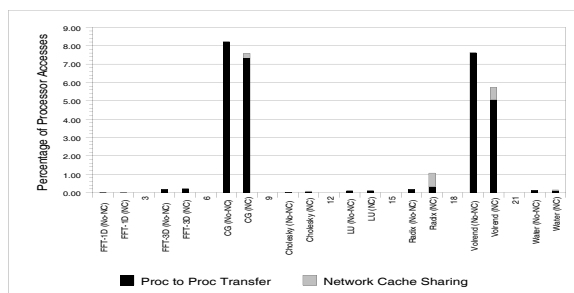
in any of the other applications, and does not appear to be an important benefit of longer network cache lines.

### 3.1.3 Intra-Cluster Sharing With and Without Network Caches

Workstation clusters with multiple processors allow co-located processes to share data inexpensively through direct processor cache-to-cache transfers. In addition, network caches may facilitate additional sharing of data subblocks by retaining data for longer periods of time than processor caches. For network cache lines longer than processor cache lines, data accesses by one processor may prefetch additional subblocks for other processors on the same cluster. Clusters of more than one processor, however, may have increased network cache conflicts due to overlapping working sets. We present measurements of these effects in the figures below.



**Figure 7 Execution Time Ratio (4 Per Cluster/1 Per Cluster)**  
Small processor caches, high network contention, 128 byte lines

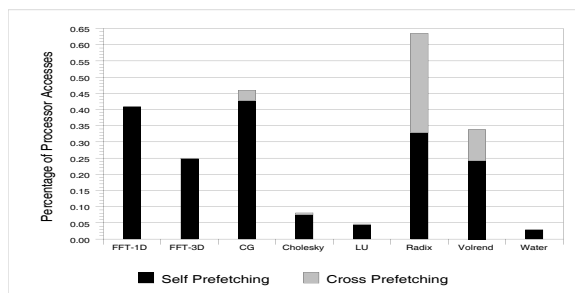


**Figure 8 Data Accesses Due to Intra-Cluster Sharing**  
Small processor caches, high network contention, four processors per cluster, 128 byte lines

Figure 7 shows the execution time ratio of an architecture of eight clusters of four processors to one of 32 clusters of single processors. In general, with or without network caches, the four-processor-per-cluster architecture performed as well the single-processor cluster architecture. Three applications (CG, Radix and Volrend) executed more than 20% faster with four processor clusters.

Figure 8 shows the percentage of processor accesses satisfied as a direct result of having more than one processor per cluster. In the architecture with network caches, processor misses can be satisfied by either direct processor-to-processor cache transfers between processor caches on the same cluster, or through the

sharing of network cache lines. In the no-NC implementation, only processor-to-processor transfers are possible. As indicated by Figure 8, the applications with the largest percentage of sharing accesses were the same ones that benefited the most from clusters of more than one processor (CG and Volrend with either architecture and Radix with network caches).



**Figure 9 Data Accesses Due to Intra-Cluster Prefetching**

Small processor caches, high network contention, four processors per cluster, 256 byte lines

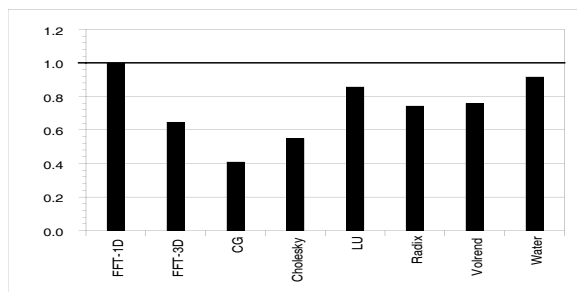
Although the network cache has the potential to increase intra-cluster sharing, Figure 8 shows that, except for one application (Radix), almost all intra-cluster sharing occurred through direct processor cache-to-cache transfers, which are also present in the no-NC architecture. Radix, on the other hand, showed a greater percentage of processor misses being satisfied by sharing in the network cache than by processor-to-processor cache transfers. Thus for Radix, architectures constructed from multiprocessor clusters were significantly beneficial only when the network interfaces contained a cache.

Figure 9 shows the percentage of processor cache accesses that were satisfied by implicitly prefetched data with 256 byte network cache lines (twice the size of the processor cache line size). The percentages for longer network cache lines were similar and have been omitted for brevity. Accesses to prefetched subblocks are categorized as *self prefetched* or *cross prefetched*, depending on the processor that fetched the network cache line. If the fetching processor and accessing processor are the same, the access is labeled self prefetching, otherwise it is labeled cross prefetching.

Similar to the results for shared lines, increased implicit prefetching due to sharing the network cache among several processors on a cluster (cross prefetching) does not appear to be a significant benefit of the network cache. Radix had a significant portion of cross prefetches (50% of all prefetches). Volrend and CG also demonstrated some cross prefetching, but to a lesser extent than Radix. The same three applications also showed some network cache line sharing, as indicated in Figure 8.

The potential for negative effects of clustering with shared network caches were only apparent in FFT-1D and Cholesky (Figure 7). The execution time of both applications increased slightly (less than 10%) with four processors per cluster and network caches. Both of these applications had increasing conflicts in the network cache when the number of processors per cluster increased, resulting in more processor evictions and a higher remote miss rate. For a more complete study of potential negative effects, see [Fletcher et al., 1994].

### 3.2 Small Caches with Low Network Contention



**Figure 10 Execution Time Ratio (NC/no-NC)**

Small processor caches, low network contention, one processor per cluster, 128 byte lines

We now consider how network caches affect performance in a system with low network contention. Figure 10 shows the relative execution time of the eight applications studied. As before, the baseline execution time is for the no-NC, one processor-per-cluster case. For all applications, the network cache provided less benefit with a low-contention network than with a high-contention network. The network cache lowers the average remote miss latency by providing more cache space on-cluster. Reducing network contention accomplishes the same goal by decreasing the time processors wait for access to the network, thereby diminishing the room for improvement. However, network caches still provided appreciable improvement for six of eight applications. The average execution time ratio of the NC to no-NC case for the low-contention network was 73%, as compared to 40% for the high-contention network. CG still showed the most improvement with an execution time ratio of 41%, as compared to 13% with high network contention (see Figure 1).

### 3.2.1 Effect of Cluster Bus Contention with Multiple Processors per Cluster

In Section 3.1.3, we observed that CG and Volrend had lower execution times with multiple processors per cluster when contention for the network was high. These benefits disappeared, however, when network contention was eliminated.

Table ?? summarizes the results for architectures built from multiple processor clusters when network contention was varied from low to high. In all cases, the total number of processors is 32. We see that for the no-NC architecture when network contention was low, multiple processors per cluster still reduced the execution time. This is not the case when network caches were used. For CG with a low-contention network and caches on the network interfaces, the four processor-per-cluster implementation ran 31% slower than the one processor-per-cluster configuration. With network contention and network caches, CG had shown a 50% execution time improvement with four processors per cluster relative to the one processor-per-cluster case. Volrend went from a 77% improvement to no improvement.

The above trends can be explained by an observed increase in cluster bus contention when contention for the network is low. Contention for the cluster bus increases as processors' remote-access stall time decreases. Thus, architectures with more than one processor per cluster will experience higher bus contention with low network contention. For CG, cluster bus utilization increased from 5% to 65% for the no-NC architecture. Adding network caches can place a higher demand on the cluster bus by reducing the average processor wait

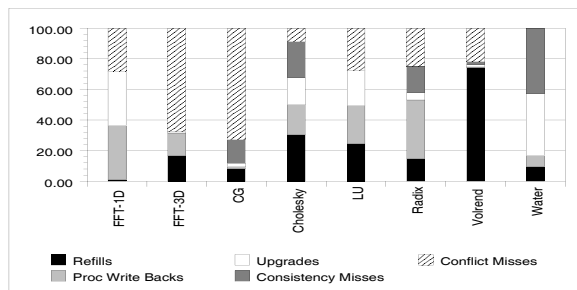
time for accesses that miss in the first-level cache. Thus, with the network caches the cluster bus utilization nearly saturated, rising from 40% to 88%. Consequently, there was still a reduction in execution time with four processors per cluster for the no-NC implementation when network contention was removed. In order for the network cache to have a larger impact on performance, the saturation of the cluster bus saturation would have to be addressed.

### 3.3 Large Processor Caches with High Network Contention

We next examine how the network cache performed when large (2 Mbyte) caches were added to each processor. Again, the cache sizes used in our simulations are scaled down to reflect the expected behavior of 2 Mbyte caches based on analysis of each application’s working set. For most applications, a large processor cache allowed the working set that was being maintained on-cluster by the network cache to fit into the processor cache. Only those applications with a significant third-level working set, or those with a second-level working set too large to fit in the processor caches, should show improvement with the addition of a network cache.

Table ?? compares the execution time effects of the addition of a network cache for the eight applications with small and large processor caches. The numbers presented are for a high contention network. Execution ratios are again presented as the execution time of the NC case over that of the no-NC case.

As Table ?? indicates, the benefit provided by the network cache decreased significantly when large per-processor caches were introduced. CG exhibited the largest decrease, with the execution time ratio of the network cache implementation rising from .13 to 1.04. This dramatic decrease in benefit occurred because the entire second-level working set fit into the large processor caches, eliminating the primary benefit provided by network caches for CG: refills of evicted processor cache lines. From Figure 11, we see that the refill category, which had been a good indicator of the effectiveness of network caches, is much lower than the levels seen in Figure 2. Figure 11 indicates that CG also exhibited a large percentage of network cache conflict misses, which resulted in increased processor cache evictions and served to further reduce the benefit offered by network caches. As before, FFT-1D did not benefit from the presence of network caches.



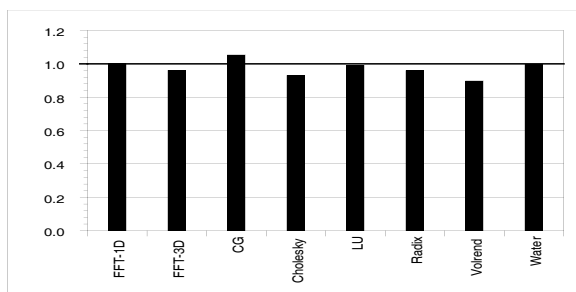
**Figure 11 Network Cache Data Accesses**

Large processor caches, high network contention, one processor per cluster, 128 byte lines

Six applications (FFT-3D, Cholesky, LU, Radix, Volrend, and Water) still showed a greater than 10% reduction in execution time for the configuration with network caches and large per-processor caches. Volrend,

LU, Cholesky, and Radix performed well because of the percentage of refill and write-back accesses that were satisfied in the network cache. Although FFT-3D had a large percentage of conflicts in the network cache, the processors did not reuse the evicted data because the matrix is transposed between each step in the algorithm. Thus, the modest number of refills and write-back accesses boosted performance over the no-NC implementation.

### 3.4 Large Processor Caches with a Low Contention Network



**Figure 12 Execution Time Ratio (NC/no-NC)**

Large processor caches, low network contention, one processor per cluster, 128 byte lines

Finally, we removed contention for the network in the architecture with large processor caches. Figure 12 shows the new execution ratios for this case. All applications showed less than 15% reduction in execution time with the addition of network caches under these circumstances. The decrease in remote miss time afforded by both the large per-processor caches and low network contention combined to reduce the benefit of the network cache.

### 3.5 Results Summary

We have presented data illustrating the effectiveness of a shared cache on the network interface under current and developing technological trends. For architectures with small processor caches and high network contention, the presence of network caches reduced execution time up to 87% over a similar architecture without network caches. Providing extra cache space to hold a larger working set was the major reason that the network cache was beneficial. Network cache lines longer than the processor cache lines were harmful for most applications, and the potential for increasing intra-cluster sharing was only realized for one application, Radix. We observed that when network contention was removed, the impact of network caches on reducing the average remote miss time decreased, resulting in a maximum improvement of 60% over the no-NC case.

Large, second-level processor caches achieve many of the same benefits as a network cache shared by all processors on a cluster. Applications where the secondary working set fit in the large processor caches saw little benefit from network caches. However, with high network contention there was still some opportunity for the network caches to reduce the average miss latency. The network cache improved the execution time for six of eight applications when network contention remained high.

Finally, we observed that a low-contention network combined with large per-processor caches decreased the performance contributions of network caches. In this case, no application experienced more than a 15% reduction in execution time with the addition of network caches.

## 4 Related Work

The concept of a multicomputer constructed from networks of workstations is not new, but has recently received renewed attention [Anderson et al., 1994]. The cluster-based system considered here most closely resembles the ASURA multiprocessing system [Mori et al., 1994]. ASURA contains clusters of processors connected by a network interface containing a cache for remote data, similar to the network cache described here. The network cache is similar to the DASH Remote Access Cache (RAC) [Lenoski et al., 1992]. The DASH RAC was designed to function primarily as a buffer to coordinate requests and replies from other clusters. STACHE [Reinhardt et al., 1994] is another form of network cache maintained in cluster memory.

Several studies have measured the impact of shared caches in multiprocessors. Nayfeh and Olukotun [Nayfeh and Olukotun, 1994] proposed a bus-based multiprocessor with shared processor caches. In their experiments, two processors per cluster performed best. Farrens et al. have proposed constructing a cluster of multiple processors sharing a second-level cache with both processors and caches constructed on the same chip [Farrens et al., 1994]. Erlichson et al. report results for a shared processor cache that is infinite and fully associative, [Erlichson et al., 1994] and compare this system to one with an individual cache for each processor. They found that when the cache access time was increased to account for multi-ported processor caches, no improvement in execution time was observed. In contrast, our results assume that each processor has an individual processor cache, and that only the network cache is shared.

There has been considerable work in the area of using data prefetching to hide long memory latencies. Both software controlled prefetching [Mowry et al., 1992], [Tullsen and Eggers, 1993] and hardware controlled prefetching [Chen and Baer, 1994], [Smith, 1982] have been studied. Hardware controlled prefetching includes methods such as longer cache lines (as is examined in this study), and specialized hardware to issue prefetch directives from runtime analysis.

## 5 Conclusions

We have evaluated the performance benefit of a shared network cache in clustered multiprocessors for current and projected processor and network technologies. By carefully choosing the simulated cache sizes to accommodate the appropriate working set for each application, we captured the expected behavior of “realistic” problems running on each configuration. We found that the network cache improved execution time for seven of eight applications when network contention was high. In general, applications for which the network cache contained a larger working set than the per-processor caches improved the most (FFT-3D, CG, Volrend, and Water), since this is the primary value of the network cache. Those applications for which the network cache held only a portion of the next larger working set (Radix and Cholesky) benefited less, and were more sensitive to the network cache line size. For FFT-1D, the network cache accommodated the

same working set as the processor caches and therefore provided no performance benefit.

Data sharing in clusters with more than one processor was found to be mainly a result of direct transfers between processor caches (CG and Volrend). Except for Radix, the network cache offered little increase in data sharing. All the applications studied experienced implicit prefetching with longer network cache line size, but only one application (CG) showed improved performance with longer lines. Although we observed some increased false sharing, false sharing was not the reason that longer network cache lines did not improve performance. Instead, it was the increased transport time associated with longer lines that overshadowed any prefetching benefit.

When we eliminated network contention by providing each processor with its own communication channel, the presence of the network cache still improved the performance of all applications except FFT-1D, but to a lesser extent than when there was high network contention. Similarly, large per-processor caches offered the most benefit when network contention was high. For CG, the secondary working set was completely contained in the larger processor caches, thereby eliminating the primary benefit of the network cache. Finally, the combined effect of a low-contention network and large per-processor caches reduced the performance benefit from a network cache implementation for these applications. This resulted from the the reduction in the average remote miss latency afforded by these two features, removing the network cache's primary contribution to performance.

The long term viability of a shared network cache depends on two factors: the extent to which per processor caches are able to hold application working sets, and the effects of network latency on performance. Large multi-megabyte second-level caches are becoming common on high end processors and may prove adequate to contain the working sets of the sort of scientific applications we studied. However, the effects of transactions processing programs or multiprogramming should be evaluated before rejecting the idea of caches located on the network interface. Database programs and multiprogrammed environments are likely to increase working set sizes. For our study, average network latency was largely dependent on network contention, and recent work suggests that network contention can be decreased substantially [Xiao and Bennett, 1995]. However, if processor speeds increase concomitant with network speeds and the bus technology can be improved to accommodate the increased bus traffic, remote miss penalty would correspondingly rise. If this were to prove to be the case, the network cache would prove to be more beneficial to performance than the results presented here for the low-contention network examples would indicate.

In summary, we found that the network cache offers substantial performance benefit when processor caches are too small to hold the application's primary working set, or when network contention limits application performance. However, these benefits diminish as processor cache size increases and network contention decreases.

## References

- [Anderson et al., 1994] Anderson, T., Culler, D., and Patterson, D. (1994). A case for networks of workstations: NOW. Technical report, University of California at Berkeley.
- [Bailey et al., 1991] Bailey, D., Barton, J., Lasinski, T., and Simon, H. (1991). The NAS parallel benchmarks. Technical Report RNR-91-002, NASA Ames.
- [Bennett et al., 1990] Bennett, J. K., Carter, J. B., and Zwaenepoel, W. (1990). Munin: Distributed shared memory based on type-specific memory coherence. In *Proceedings of the 2nd ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming*, pages 168–176.
- [Bennett et al., 1995a] Bennett, J. K., Fletcher, K. E., and Speight, E. (1995a). Classification of data accesses to shared remote data caches in cluster multiprocessors. Technical Report TR-9502, Rice University.
- [Bennett et al., 1995b] Bennett, J. K., Fletcher, K. E., and Speight, W. E. (1995b). Cache size choices for simulation of cluster-based multiprocessors. Technical Report TR-9507, Rice University.
- [Bershad et al., 1993] Bershad, B., Zekauskas, M., and Sawdon, W. (1993). The Midway distributed shared memory system. In *Proceedings of COMPCON93*, pages 528–537.
- [Blumrich et al., 1994] Blumrich, M. A., Li, K., Alpert, R., Dubnicki, C., Felten, E. W., and Sandberg, J. (1994). Virtual memory mapped network interface for the SHRIMP multicomputer. In *Proceedings of the 21st Annual International Symposium on Computer Architecture*, pages 142–153.
- [Boyle et al., 1987] Boyle, J., Butler, R., Disz, T., Glickfeld, B., Lusk, E., Overbeek, R., Patterson, J., and Stevens, R. (1987). *Portable Programs for Parallel Processors*. Holt, Rinehart and Winston, Inc.
- [Chaiken et al., 1991] Chaiken, D., Kubiawicz, J., and Agarwal, A. (1991). Limitless directories: A scalable cache coherence scheme. In *Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 224–234.
- [Chen and Baer, 1994] Chen, T.-F. and Baer, J.-L. (1994). A performance study of software and hardware data prefetching schemes. In *Proceedings of the 21st Annual International Symposium on Computer Architecture*.
- [Dubois et al., 1993] Dubois, M., Kkeppstedt, J., Ricciulli, L., Ramamurthy, K., and Stenström, P. (1993). The detection and elimination of useless misses in multiprocessors. In *Proceedings of the 20th Annual International Symposium on Computer Architecture*, pages 88–97.
- [Dwarkadas et al., 1994] Dwarkadas, S., Jump, J. R., and Sinclair, J. B. (1994). Execution-driven simulation of multiprocessors: Address and timing analysis. In *Journal of Transactions on Modeling and Computer Simulation*.
- [Erlichson et al., 1994] Erlichson, A., Nayfeh, B. A., Singh, J., Olukton, K., and Hennessy, J. (1994). The benefits of clustering in shared address space multiprocessors: An applications-driven investigation. Technical Report CSL-TR-94-632, Computer Systems Laboratory, Stanford University.
- [Farrens et al., 1994] Farrens, M., Tyson, G., and Pleszkun, A. R. (1994). A study of single-chip processor/cache organizations for large numbers of transistors. In *Proceedings of the 21st Annual International Symposium on Computer Architecture*, pages 338–347.
- [Fletcher et al., 1994] Fletcher, K. E., Speight, W. E., and Bennett, J. K. (1994). Techniques for reducing the impact of inclusion in shared network cache multiprocessors. Technical Report TR-9413, Rice University.
- [Keleher et al., 1994] Keleher, P., Dwarkadas, S., Cox, A., and Zwaenepoel, W. (1994). Treadmarks: Distributed shared memory on standard workstations and operating systems. In *Proceedings of the 1994 Winter Usenix Conference*, pages 115–131.
- [Kuskin et al., 1994] Kuskin, J., Ofelt, D., Heinrich, M., Heinlein, J., Simoni, R., Gharachorloo, K., Chapin, J., Nakahira, D., Baxter, J., Horowitz, M., Gupta, A., Rosenblum, M., and Hennessy, J. (1994). The Stanford FLASH multiprocessor. In *Proceedings of the 21st Annual International Symposium on Computer Architecture*, pages 302–313.
- [Lenoski et al., 1992] Lenoski, D., Laudon, J., Gharachorloo, K., Weber, W., Gupta, A., Hennessy, J., Horowitz, M., and Lam, M. S. (1992). The Stanford DASH multiprocessor. *Computer*, pages 63–79.

- [Li, 1988] Li, K. (1988). Ivy: A shared virtual memory system for parallel computing. In *Proceedings of the 1988 International Conference on Parallel Processing*, pages 94–101.
- [Mori et al., 1994] Mori, S., Saito, H., Goshima, M., Yanagihara, M., Tanaka, T., Fraser, D., Joe, K., Nitta, H., and Tomita, S. (1994). A distributed shared memory multiprocessor: ASURA - memory and cache architectures. In *Proceedings of the 1994 ACM International Conference on Supercomputing*, pages 740–749.
- [Mowry et al., 1992] Mowry, T., Lam, M. S., and Gupta, A. (1992). Design and evaluation of a compiler algorithm for prefetching. In *Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 62–73.
- [Nayfeh and Olukotun, 1994] Nayfeh, B. A. and Olukotun, K. (1994). Exploring the design space for a shared-cache multiprocessor. In *Proceedings of the 21st Annual International Symposium on Computer Architecture*, pages 166–175.
- [Papamarcos and Patel, 1984] Papamarcos, M. and Patel, J. (1984). A Low Overhead Coherence Solution for Multiprocessors with Private Cache Memories. In *Proceedings of the 11th International Symposium on Computer Architecture*, pages 340–347. IEEE.
- [Press et al., 1988] Press, W. H., Flannery, B. P., Teukolsky, S. A., and Vetterling, W. T. (1988). *Numerical Recipes in C*. Cambridge University Press, Cambridge.
- [Reinhardt et al., 1994] Reinhardt, S., Larus, J., and Wood, D. (1994). Tempest and typhoon: User-level shared memory. In *Proceedings of the 21st Annual International Symposium on Computer Architecture*, pages 325–337.
- [Rothberg et al., 1993] Rothberg, E., Singh, J. P., and Gupta, A. (1993). Working sets, cache sizes, and node granularity issues in large-scale multiprocessors. In *Proceedings of the 20th Annual International Symposium on Computer Architecture*, pages 14–25.
- [Schoinas et al., 1994] Schoinas, I., Falsafi, B., Lebeck, A. R., Reinhardt, S. K., Larus, J. R., and Wood, D. A. (1994). Fine-grain access control for distributed shared memory. In *Proceedings of the Sixth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 297–306.
- [Smith, 1982] Smith, A. J. (1982). Cache memories. *Computing Surveys*, 14.
- [Speight et al., 1994] Speight, W. E., Fletcher, K. E., and Bennett, J. K. (1994). Working set requirements and performance of network caches in cluster-based multiprocessors. Technical report, Rice University, TR 9412.
- [Tullsen and Eggers, 1993] Tullsen, D. M. and Eggers, S. J. (1993). Limitations of cache prefetching on a bus-based multiprocessor. In *Proceedings of the 20th Annual International Symposium on Computer Architecture*, pages 278–288.
- [Weber, 1993] Weber, W.-D. (1993). *Scalable Directories for Cache-Coherent Shared-Memory Multiprocessors*. PhD thesis, Stanford University.
- [Woo et al., 1995] Woo, S. C., Ohara, M., Torrie, E., Singh, J. P., and Gupta, A. (1995). Methodological considerations and characterization of the SPLASH-2 parallel application suite. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pages 24–36.
- [Xiao and Bennett, 1995] Xiao, Y. and Bennett, J. K. (1995). Performance of multi-channel networks in clustered multiprocessors. Technical Report TR-9510, Rice University.