The Effectiveness of SRAM Network Caches in Clustered DSMs

Adrian Moga and Michel Dubois

CENG 97-11

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, California 90089-2562
(213) 740-4475

July 1997

The Effectiveness of SRAM Network Caches in Clustered DSMs

Adrian Moga and Michel Dubois

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, CA 90089-2562
(213)740-4475
Fax: (213) 740-7290
{moga,dubois}@paris.usc.edu

USC Computer Engineering Technical Report 97-11

July 1997

Abstract

The frequency of accesses to remote data is a key factor affecting the performance of all Distributed Shared Memory (DSM) systems. Remote data caching is one of the most effective and general techniques to fight processor stalls due to remote capacity misses in the processor cache. The design space of remote data caches (RDC) has many dimensions and one essential performance trade-off: hit ratio versus speed. Some recent commercial systems have opted for large and slow (S)DRAM network caches (NC), but others completely avoid them because of their damaging effects on the remote/local latency ratio.

In this paper we will explore small and fast SRAM network caches as a means to reduce the remote stalls and capacity traffic of multiprocessor clusters. The major appeal of small fast NCs is that, by handling conflict misses and some capacity misses, they reduce the page relocation rate of main memory page caches, which satisfy the bulk of capacity misses. To maximize this benefit for a large spectrum of application behavior we propose to organize the NC as a victim cache for remote data. We also propose a novel and scalable method to control the page cache by integrating page relocation mechanisms into the network victim cache.

Our results, obtained through detailed simulations for eight SPLASH2 benchmarks, indicate that SRAM NCs and page caches outperform DRAM NCs for applications with large spatial locality and regular access patterns. In these cases, both SRAM NCs and the page cache can be accessed faster than a DRAM NC, thus expediting the processing of remote misses. The large spatial locality avoids fragmentation in the page cache, yielding a better hit ratio. When spatial locality is low and access patterns are irregular, DRAM NCs are still better. In this latter case, our proposed victim cache outperforms other systems with page caches. Our method for integrating the page cache control mechanisms into the network cache compares favorably to previously proposed methods.

Keywords: remote data caching, DSM clusters, coherence protocols, NUMA, performance evaluation.

1. Introduction

In a DSM system the latency of remote data accesses can be from several times to an order of magnitude higher than that of local data accesses. Processor caches capitalize on the temporal locality of memory accesses and significantly alleviate the high latency problem. However, speed and/or cost considerations limit their size. Consequently, in large applications, the remote working set overflows the cache, and capacity misses to remote data generate large access penalties.

Several techniques have been applied at various system levels to cut the number of remote capacity misses. Techniques available to the programmer and compiler, such as blocking [11], and to the operating system, such as page placement [13] migration and replication [22][23], are effective up to a point. Further improvements are attained by caching remote data in local memory resources acting as backups for the processor caches. This more powerful approach eliminates the hassle of dealing with the problem in software. One way or another, practically all commercially available (KSR-1 [9], NUMA-Q [16], Exemplar [1]), prototype (DASH [15], S3.mp [18]), or proposed (DDM [5], COMA-F [6], Simple COMA [21], R-NUMA [3]) DSM architectures have provisions for remote data caching. A notable exception is the SGI Origin [14] which relies exclusively on page migration and replication.

The design of a cache for remote data involves several key decisions. A fundamental choice is whether to allocate separate resources for caching local and remote data or to cache both in the same memory as is done in COMA [5] [6]. A study [25] has found a slight performance advantage for systems with large dedicated remote data caches (RDCs). In this paper, we focus on dedicated remote data caches (RDCs). The size of the RDC impacts the choice of the memory technology (DRAM or SRAM) and its speed. There is a clear performance trade-off between the effect on the number of remote capacity misses (dependent on the RDC size) and the effect on remote access latencies (dependent on the speed). The RDC can be organized as a network cache [15][18][16][1] or as a page cache [21][20] managed under various policies. Finally, the bus protocol in a multiprocessor cluster also affects the RDC design.

The contribution of this paper is to explore the design and evaluate the performance of small and fast network caches (NCs) in the context of clustered DSMs. The small NCs are built in the same technology as the processor caches and should be convenient to use. We present and evaluate several techniques that allow a cluster to achieve the same low remote capacity miss ratio as if equipped with a much larger NC, but without paying the penalty of slower remote accesses. Essentially, we propose to organize the NC as a victim cache for remote data and to expand its capacity with a page cache in main memory. We also propose a novel mechanism to control the page cache in the NC. Our results for eight SPLASH2 benchmarks show that such techniques for caching remote data outperform large NCs for applications with regular access patterns and large spatial locality. For applications with irregular access patterns, and sparse and large remote working sets, our proposal to use a victim NC leads to a reduction of data traffic and page relocation overheads with respect to other systems with page caches. Our proposed method of implementing the control mechanisms for the page cache adds scalability to previous proposals and saves memory without loss of performance.

The rest of this paper is organized as follows. In the next section we will present the background and motivation. Issues and solutions in the design of small network caches are detailed in Section 3.The performance model and the evaluation methodology follow in Sections 4. and 5. The results are discussed in Section 6. We end with a discussion of related work and with our conclusions.

2. Background and Motivation

To avoid possible confusions, we first introduce some terms. The processor caches are simply called "caches". The memory resources dedicated to caching remote data are collectively called "remote

data caches" or "RDC". The network cache (NC) and the page cache (PC) are examples of RDCs. The terms "miss ratio" and "misses" refer to the processor caches, unless otherwise specified. We refer to all the remote (inter-cluster) coherence and cold misses as "necessary misses".

In Figure 1 we show two organizations for a DSM cluster. In both cases, a number of processors with their own cache are connected by a bus. A pseudo-processor (PP) [15] acts as a representative of all the caches outside the cluster and supports system-level coherence as an extension of the bus multiprocessing protocol. The PP also controls a network cache whose purpose is to reduce costly accesses over the network caused by capacity misses in the caches.

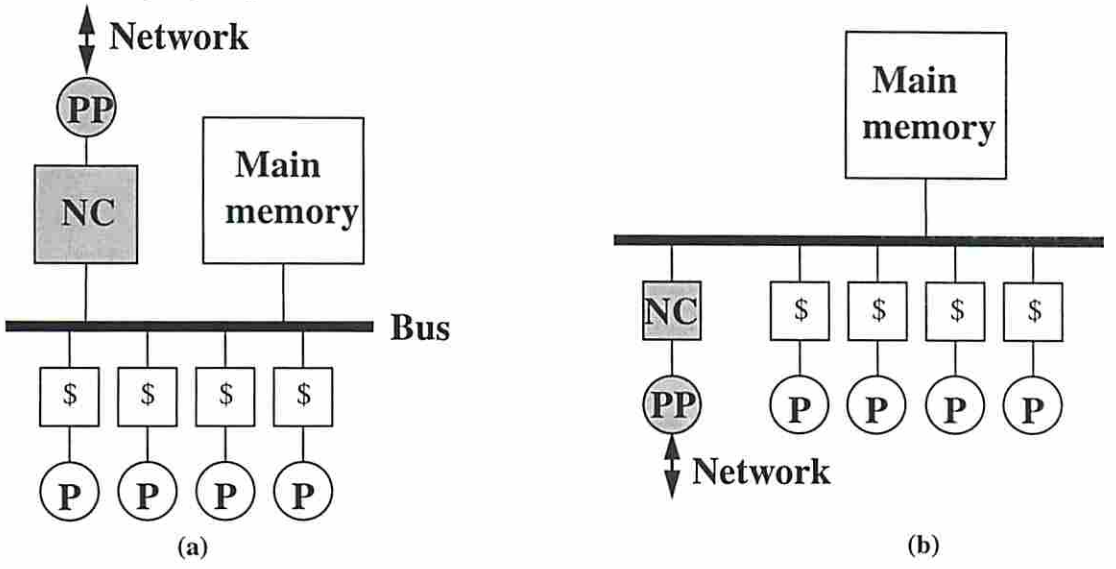

Figure 1. Two ways of incorporating a network cache in a DSM cluster

Figure 1(a) illustrates a large NC, designed as another level in the cache-memory hierarchy. Although a large NC has the potential to satisfy many capacity misses, it is made of DRAM and its slow access time is in the critical path of *all* misses to remote data. On an NC miss, the NC delays not only the issuing of a request to the home node, but also the time when a reply can be sent, because of additional NC accesses in other nodes [14]. The penalty of accessing a slow NC may be negligible when remote latencies are high. However, in recent commercial systems such as the SGI Origin [14] and Convex Exemplar [1] the ratio of remote versus local memory access times is less than three and thus any additional DRAM access can significantly affect this ratio. This has prompted the designers of the Origin to abandon the NC and to rely exclusively on page migration and replication [23] to improve data locality.

The NC can also be small and fast and act as a peer of the caches, as shown in Figure 1(b). As such, the NC quickly snoops all bus transactions and the overheads in remote accesses are virtually eliminated. However, a small NC has a lower hit ratio than a big NC, which is a serious handicap. Current commercial systems seem to require 32-128 MB of network cache, which is prohibitive in SRAM technology.

One way to improve the hit ratio of small NCs is to reduce the number of references that must access it, which is accomplished by redirecting some of them to the local main memory. Page caches [21][3] residing in the cluster's main memory can hold replicas of remote pages aliased under local addresses. With the support of tags for fine-grain sharing, a page cache can replicate even pages that are actively read and written by several processors, unlike operating system controlled page replication. Thus, the page cache can be seen as a means to extend the capacity of a small NC. The first proposal for a system incorporating both page and network caches was R-NUMA [3], where decisions on page relocation are based on capacity miss counters maintained in memory, one for each cluster-page pair.

Three questions must be answered before the combination of page caches and SRAM NC can be declared viable. The first question, whether page caches need any NC at all to work at their best, was answered in [3]. The evaluation of R-NUMA shows significant improvements of the execution time for some applications in the presence of a 32KB network cache. As we will demonstrate later, the NC is also instrumental in satisfying conflict misses and in shielding the page cache management policies from their noise. However, questions about the appropriate size and method of integration of the NC into the cluster are not answered. In particular it is doubtful that either a 128 byte NC, as proposed by R-NUMA, is sufficient, or that a larger cache, four times the size of the processor caches, is realistic [3].

A second question, still open, is whether page cache and small NCs together can outperform large NCs. For a particular workload and a certain amount of memory dedicated to caching remote data, Figure 2 illustrates possible design points and their estimated remote stall. If a small amount of RDC memory is sufficient, one would clearly prefer an SRAM NC. When the amount of memory required exceeds a threshold caused by cost-effectiveness, packaging or power considerations, some or all of the RDC must be in DRAM. With a DRAM NC there is an additional penalty for searching the NC on all cache misses to remote data. By contrast a page cache in main memory can extend the capacity of the SRAM NC with no extra penalty on the necessary cache misses, but usually with a lower hit ratio than a cache with fine-grain allocation. In any case, DRAM NC or page cache, the size of the RDC can only be increased up to the point where all capacity misses are satisfied and the remote stall saturates. At this point, the size of the page cache is usually bigger than that of the DRAM NC, because of fragmentation due to the larger grain of allocation. The page cache solution is guaranteed to outperform the large NC at saturation, but, in some cases, the corresponding page cache size may be prohibitive. Depending on the actual size of the RDC, system-specific parameters, and application characteristics either design could prevail.

Page caches have trouble with a class of applications having large and sparse remote working sets, with low spatial locality. For these applications the utilization of the page cache is poor and the overhead of page relocation is high, sometimes leading to page cache thrashing. Thus, a third question is whether particular designs of a small NC can optimize the performance of the page cache. The improvements of the page cache performance have to be measured along three axes: the remote stall, the page relocation overhead, and the page cache size. As we will show, there are features of the NC that promote reductions of the page relocation overhead, of the page cache size, and of the remote stall.

These three questions call for an evaluation of various organizations of small and fast NCs with or without page caches. In particular we propose to organize the NC as a victim cache [7]. In this context we offer an alternative to R-NUMA's page relocation counters. The R-NUMA solution lacks scalability because it only works with full-map, centralized directories and, for every page, it requires a number of counters equal to the number of clusters.

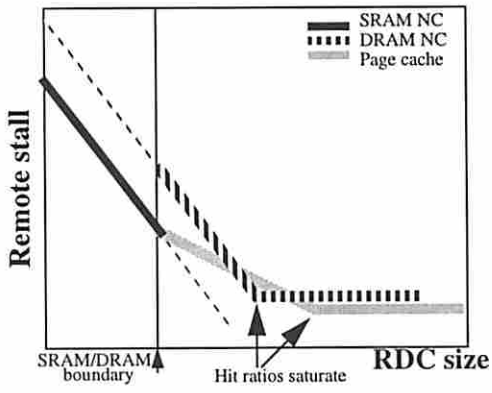

Figure 2. Qualitative impact of the RDC design and size on the remote stall

3. Design Issues and Solutions

It is imperative that the utilization of a small NC be maximized. This can be partly accomplished by breaking the inclusion property between the caches and the NCs and by improving the bus protocol. Also important is the extension of the NC with other local memory resources, such as a page cache in main memory. We discuss these issues and offer some solutions next.

3.1. The inclusion property

Systems with large DRAM NCs, such as the Sequent Sting [16], usually provide inclusion between the caches and the NC. This property may be required by the inter-cluster coherence protocol (SCI for Sting) or may be easier to implement, but it is certainly a waste of resources. Under inclusion frames are allocated both in the cache and in the NC on every cache miss. Besides the redundant allocation of resources, the NC frame is useless whenever an invalidation for the block is received before the cache frame is replaced. Additionally, conflicts in the NC can degrade the cache hit ratio. Blocks victimized in the NC because of conflicts are forcefully evicted from the caches, in order to maintain inclusion. In some cases, this may lead to better performance when the NC is bypassed. To eliminate this effect, the NC must have a size greater or equal to the sum of all the caches in the cluster [2].

Inclusion can be relaxed for clean blocks, as was done in [4] and [3]. In this case, the NC allocates a frame at the time of a cache miss, but, when a clean block is victimized in the NC, it is not invalidated in the caches. By relaxing inclusion for clean blocks, the utilization of the NC is improved and the size of the NC can be reduced.

For even better resource utilization, the NC can host only the lines victimized by the caches. Such lines have a much better chance of later incurring a capacity miss, which the NC could satisfy. By delaying the moment when a frame is allocated in the NC, we can expect a better hit ratio for the NC, especially when its size is small. Also, the cache hit ratio can no longer be affected by capacity and conflicts misses in the NC and the overall performance cannot be worse than that of a system without NC.

3.2. The bus protocol

The bus protocol must be enhanced to integrate a victim cache for remote data. Under the Illinois protocol (MESI) [19] or its extension, the MOESI protocol, it is not possible to save a clean block when it is replaced, since the block does not reach the bus. Whereas this capability is pointless for blocks residing in the local memory, it may be advantageous in a NUMA environment to save a clean remote block in a victim cache when it is the last copy in the node. The victim cache has no problem capturing dirty remote blocks, as they are written back when victimized.

We propose a simple extension to the MESI protocol. The protocol MESIR includes a new state R, denoting mastership for a remote clean block. This state is similar to the Shared state, except for the actions following its victimization. When a remote block is first brought clean into a node, it is cached in state R. Other processors in the node can later acquire Shared copies through cache-to-cache transactions. When victimized, a block in state R generates a replacement transaction. If other caches have the block in state Shared, one of them assumes mastership and changes to state R. Otherwise, the node's victim cache accepts the block.

Another feature of MESI protocols (including MESIR) may affect the performance of the victim cache. When a dirty (M) block is downgraded to a Shared state, the memory must be updated to keep it clean. For remote blocks, these remote memory updates can be a serious overhead. This problem was solved in the DASH [15] by a Remote Access Cache to capture these remote write-backs. A victim cache

for remote data performs the same service, but the write-backs pollute the cache because they force the allocation of a frame in the victim cache while the caches still have a copy. The problem could be solved by adding an explicit dirty-shared (O) state in the protocol. However our evaluations have indicated very little benefit for doing so.

3.3. Incorporating a page cache

Page caches for remote data are an attractive way to reduce the number of remote capacity misses. Initially proposed for Simple COMA [21], the page cache holds remote data aliased under local addresses. It can be stored in main memory, just like local data. Items are allocated in the page cache at the granularity of the system's page, because the aliasing mechanism relies on the virtual memory support (MMU). However, coherence is maintained at a finer grain (usually that of a cache block). The coherence protocol uses global identifiers and needs tables to translate them to and from the local aliases.

Relocating a remote page to the page cache is a costly operation, requiring several complex actions on the part of the operating system. The key is to offset this cost by satisfying enough remote capacity misses during the lifetime of the entry in the page cache. The policies for the management of the page cache in Simple COMA have proven very inefficient for certain applications. Because every reference to a remote page absent from the page cache generates a relocation, the page cache can easily thrash. More recently, R-NUMA [3] has improved the relocation policy by adding counters for capacity misses to the directory. Each counter is associated with a page-processor pair. A page is only relocated when its counter exceeds a threshold. This has a double effect. First, the relocation policy is solely based on remote capacity misses (as opposed to all remote misses, including coherence), which is a refinement. Second, if page cache thrashing is a problem, the page counters scale it down by a sampling effect.

A victim cache can improves the behavior of the page cache in three ways. First, by cutting down on conflict misses, it reduces the noise in the activity of the page counters for capacity misses. Second, for applications with large and sparse remote working sets. such as Radix, the organization of the NC can help reduce the page relocation activity and the amount of page cache thrashing. If the NC is set-associative and indexed with the least significant bits of the page number, blocks from the same page map into the same set, and each set is an intermediate storage for blocks from a remote page. This prevents the counters from triggering a relocation even when only a couple of blocks in a page generate capacity misses. Finally, with this organization of the NC, it is attractive to associate the relocation counters directly with the sets of the victim cache. This significantly cuts down the number of counters. Also, the concentration of controls and counters in the same place facilitates the monitoring of the page cache and the incorporation of feedback to dynamically adjust the relocation thresholds and/or the page cache size. An adaptive threshold scheme will be explained later. The design of a victim cache with relocation counters is detailed next.

3.4. Network Victim Caches and Page Relocation Counters

As explained before, the allocation of a page to the page cache of a certain cluster is triggered in R-NUMA by a counter for capacity misses, associated with the particular page and cluster, a scheme which lacks scalability. Although these counters provide accurate information about the most deserving candidates for relocation, a 256 clusters system, for example, requires 256 counters (bytes) for every 4KB page, a 6.67% memory overhead. Very little of this memory is actually used because many pages are never touched by many clusters. Besides, this memory could be better used to expand the page cache and have less need to be picky about what pages to relocate. There are further disadvantages. In order to distinguish capacity misses from necessary misses, R-NUMA relies on the presence bits in the directory. When a cluster requests a block and the presence bits indicate it should already have it, a capacity miss is counted. In a non-notifying protocol, the presence bits already carry this information for clean blocks. Additionally, R-

NUMA requires the bits to remain turned on after a dirty block is written back. Although the modification is feasible, it introduces more false invalidations.

As an alternative to R-NUMA's relocation counters, we can instead monitor the activity in the victim cache. An important observation is that every capacity miss is preceded by a victimization at some level of the memory hierarchy. Although there is not a one-to-one correspondence between the two (there are more victimizations than capacity misses), it is worth trying a relocation policy driven by victimization counts. A network victim cache is the ideal place to count such events. Moreover, when its sets are indexed by the page address, the counters could be attached directly to the sets, although there is a chance that more than one page will share a counter. These counters are scalable and do not interfere with the coherence protocol at the directory. It is true that counter sharing can be implemented even with the original proposal. However, when a decision must be taken about what page to relocate, the victim cache has implicit candidates, as indicated by the address tags in the corresponding set. The directory-controlled counters would have to store and manage such information explicitly. The robustness of counter sharing is something well worth investigating, but beyond our scope here. We only point out that, since relocation is handled in software, there is great flexibility in preventing worst case scenarios. Another appeal of our relocation mechanism is that it does not require a full-map directory implementation. As such, even systems based on limited pointer or linked lists protocols (like NUMA-Q) could make efficient use of the page caches.

We will evaluate a network cache where the sets are indexed by the page address and counters are incremented after every victimization in the NC. When a counter exceeds a threshold, the predominant tag for the frames in the set indicates the page to relocate. This information can be easily computed by the software relocation handler given access to the tags or can be provided by specialized hardware. The policy can be improved by decrementing the counters when invalidations are received. This should be done only if none of the processor caches or the NC holds the invalidated block. The justification is that, if the block was previously victimized from the cluster, a counter was incremented, but the late invalidation indicates the next miss will be a coherence miss, so the count can be corrected. Our base system does not use this improvement, because we have not observed that it is not significant.

Event	No NC	DRAM NC	SRAM NC	SRAM NC & PC
PC hit	-	-	-	DRAM access
PC miss	-	-	-	Remote access
NC hit	-	DRAM access +tag checking	cache-to-cache transfer	cache-to-cache transfer
NC miss	Remote access	Remote access +tag checking	Remote access	Remote access

Table 1: Latency components for remote data references

4. Performance Model

The events monitored for the performance model are the cache misses to remote data (i.e. where the home node is not the local node). In different system organizations, these misses complete in different places and with different latencies. Table 1 indicates the components of these latencies for several system configurations, and the possible outcomes for each cache miss. The systems can be equipped with an NC and a page cache (PC). The estimate of performance is the total remote **read** stall RS:

$$RS = N_{NC}^{hit} L_{NC}^{hit} + N_{NC}^{miss} L_{NC}^{miss} + N_{PC}^{hit} L_{PC}^{hit} + N_{PC}^{miss} L_{PC}^{miss} + N_{PC}^{rel} T_{PC}^{rel} \quad (1)$$

N stands for the count of events and L for their latency. In addition, the average overhead of relocating pages to the page cache is given by T^{rel}. This overhead includes the interruption of the processor, the execution of the software handlers, and the re-mapping of the page (TLB shootdown). This model does not account for contention and uses a constant, average value for latencies when, in fact, two- and three-hop transactions have different latencies. In each access to the DRAM NC the block and its address tag and state bits are fetched simultaneously. Then its tag is checked and the appropriate block in the set is multiplexed onto the data bus. In the case of the page cache a two-bit block state is kept in SRAM and is snooped at bus speed.

5. Evaluation Methodology

5.1. System configurations

We evaluate systems consisting of eight nodes with four processors each, for a total of 32 processors. The caches are write-back. In the base configuration, the caches are 16KB, two-way set-associative, with 64-byte blocks and LRU replacement. Variations of the associativity will be explored.

We look at three systems with SRAM NCs. The NC has the same size as the caches and is always four-way set associative. In one configuration called *nc* inclusion is relaxed for clean blocks. The other two configurations correspond to the victim cache organization, with set indexing based on the block address (*vb*) or the page address (*vp*). Whenever showing the estimated remote stall, we include results for a configuration with an infinite SRAM NC (*ncs*), such that the directory handles only necessary misses.

Three other systems (*ncp, vbp, vpp*) are obtained by adding page caches to *nc, vb*, and *vp*. Evaluations are done for different page cache sizes, expressed as a fraction of the total data set size for the application. For example, *ncp5* indicates that the page cache size is one fifth of that of the application data. The replacement policy for the page cache is least recently missed [3]. All three systems use per-page per-cluster relocation counters controlled by the directory. A fourth system, *vxp*, derived from *vpp*, integrates the page cache control mechanisms (per-set victimization counters) into the victim cache.

We also compare systems with page caches to a system with a 512KB DRAM NC (*NCD*). In this case, the page cache size is also 512KB. We picked 512KB based on the assumption that network caches are eight times larger than the entire amount of processor cache in the cluster. This holds true for NUMA-Q [16], for example.

The times taken by the components of the latency listed in Table 1 are shown in Table 2. They are expressed in cycles for a 100MHz bus, because processor speeds are irrelevant (except for the page relocation overhead which has software components).

Event	Latency (10ns. bus cycles)
DRAM access	10
Tag checking	3
Cache-to-cache transfer	1
Remote access	30
Page relocation	225

Table 2: Latencies for the events in Table 1

5.2. Evaluation approach

We derive our results using trace-driven simulation. Eight SPLASH-2 [24] benchmarks using a single-address space programming model have been compiled for a SPARC V7 architecture. They are listed in Table 3. Data is placed in main memory using a first-touch policy [17]. The SPLASH-2 benchmarks are optimized, so that this policy is close to being optimal in minimizing the number of remote accesses. We have done a small modification to LU. With the original LU code, the first touch placement resulted in all pages been local for cluster 0, because the master processor initializes the matrix in the parallel section.

Benchmark	Parameters	Shared memory (MB)
Barnes	16K bodies	3.94
Cholesky	tk15.O	21.37
FFT	64K points	3.54
FMM	16K bodies	29.23
LU	512 x 512	2.16
Ocean	258 x 258	15.52
Radix	1M integers	9.87
Raytrace	car	34.86

Table 3: Characteristics of the benchmarks

6. Simulation Results

6.1. Effects of the network cache on the cluster miss ratios

6.1.1. Miss ratio reduction by the victim cache

To understand the extent of capacity and conflict misses to remote data for different cache organizations and to show the effectiveness of the victim cache, we plot cluster miss ratios in Figure 3. For direct-mapped, two-way, and four-way set-associative processor caches, the miss ratios are shown with no NC all, and with victim NC of sizes 1KB and 16KB.

The 1KB victim cache in the configuration with two-way set-associative caches is effective in maintaining the node miss ratio at the same levels as four-way set-associative caches with no NC, but it is too small to satisfy all the conflict misses in direct-mapped caches, especially in LU, Cholesky, and FMM.

The 16KB victim cache further reduces the node miss ratio, clearly helping not only with conflict misses, but also with some capacity misses. The capacity misses satisfied by the victim cache come in different amounts and forms for the eight applications. The amount of capacity misses satisfied by the victim cache can be higher if the applications's cache miss ratio curve has a knee close to the 16KB point, as is the case in Barnes and Ocean. Radix has a predominant reduction in write capacity misses, while all other applications have fewer read capacity misses.

To inject moderate amounts of conflict misses in our evaluations, we have decided to use two-way

set-associative caches for the base configurations from now on. This way, the evaluated systems should behave like realistic systems, with four-way set-associative caches, but also with larger data set sizes.

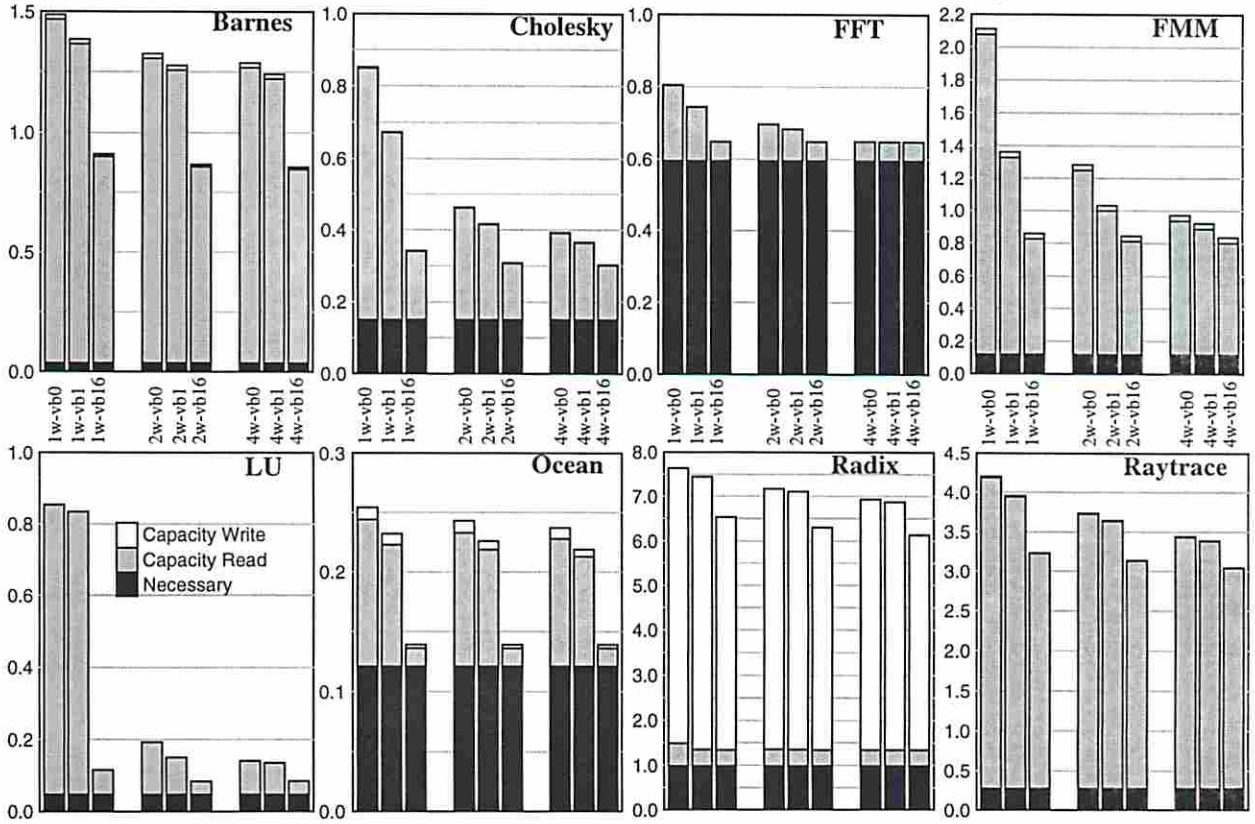

Figure 3. Effects of the network victim cache on the cluster remote miss ratio (%)
The miss ratio is in percentage of all shared (non-stack) references. The processor caches are 16KB with set-associativities of 1,2, and 4 (1w,2w,4w). The network cache sizes are 0, 1KB, and 16KB (vb0, vb1, vb16). The NC is always four-way set-associative and organized as a victim cache.

6.1.2. Effects of inclusion

To understand the benefits of a network victim cache as opposed to a NC which maintains inclusion for dirty blocks only, Figure 4 compares cluster miss ratios.

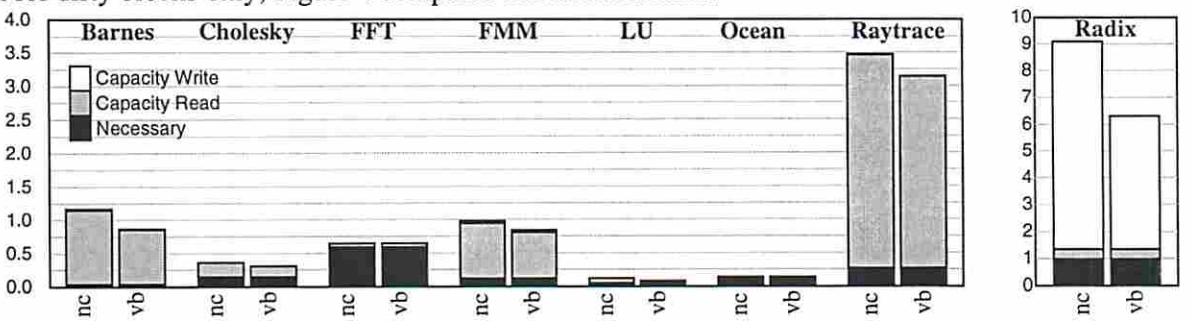

Figure 4. Cluster miss ratios (%) for different ways of integrating the network cache into the cluster
The NCs are 16KB, four-way set-associative. *nc* maintains inclusion for dirty blocks; *vb* is a victim cache.

Clearly, miss ratios are smaller for the system with victim caches. This is due to the better utilization of the NC, which never replicates information already present in one of the caches. Thus, the victim NC offers additional capacity to the node. The reduction of the miss ratio is moderate in applications with

mostly read capacity misses (Barnes, FMM, LU, Raytrace), but quite impressive in the presence of write capacity misses, as in Radix. In this latter case, *nc* exhibits side effects from maintaining inclusion for dirty blocks. The NC can accommodate fewer dirty blocks than the ensemble of processor caches and becomes a limiting factor for the amount of remote data that the cluster can hold. Even worse is the increase in the write-back traffic accompanying this effect. The conclusion is that maintaining any kind of inclusion for such small NCs is something to avoid, if performance improvements are expected over a large spectrum of applications.

6.1.3. Effects of the victim cache indexing scheme

In Figure 5 we show cluster miss ratios for two ways of indexing the victim cache, one based on the least significant bits of the block address (*vb*) and the other based on the least significant bits of the page address (*vp*). Although indexing the victim cache with page address bits may seem dangerous because many applications have large spatial locality the results actually indicate improvements in *vp* for FMM and Radix, two applications with little spatial locality and irregular access patterns. As expected, applications with large spatial locality (Cholesky, Ocean) exhibit degradation of the cluster miss ratio because of conflicts in the NC. However, this degradation can never lead to results worse than when no NC is present because inclusion is not maintained

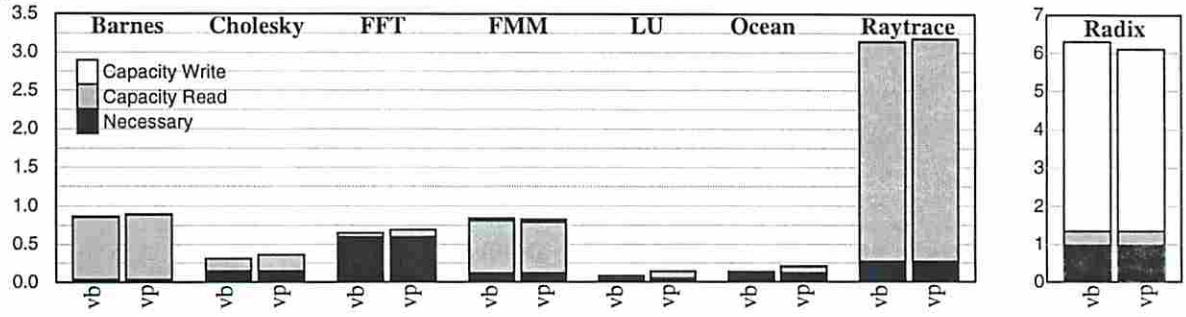

Figure 5. Cluster miss ratios (%) for different ways of indexing the victim cache
The network victim caches are 16KB, four-way set-associative. *vb* uses the least significant bits (LSB) of the block address to compute the set number; *vp* uses the LSB of the page address.

6.2. Effects of the victim cache on systems with page caches

The performance benefit of a page cache depends on the relocation threshold. It is unclear whether the same value of the threshold, as done in other studies [3], leads to a fair comparison of systems, and even applications; therefore we have designed an adaptive threshold policy. This policy uses thresholds that can be independently tuned for each node. The thresholds are initialized to 32 and incremented by 8 every time thrashing is detected in the page cache. The mechanism for thrashing detection is based on hit counters for the page cache frames. These counters are saturating and are maintained by the hardware. Threshold adjustments are done in software by the page relocation handlers. When a page cache frame is reused, the hit count is adjusted by subtracting the *break-even count*, a number considered to be the minimum hit count to offset the costs of page relocation. The result is accumulated in another counter, the *thrashing indicator*. If the thrashing indicator is negative after a certain number of frame reuses, called the *monitoring window*, the relocation threshold is incremented and all the hit counters are reset. We have used a break-even count of 12 and a monitoring window of size twice the number of frames in the page cache. This policy has proven effective in reducing page cache thrashing. More sophisticated policies, with faster adaptive response are possible, but beyond the scope of this paper.

In Figure 6 we compare the adaptive policy to the policy with a fixed threshold of 32 throughout

the execution of the application when the page cache size is 1/5. Thrashing is successfully handled by the adaptive policy for Barnes and Radix. With smaller page caches, thrashing occurs in other applications as well.

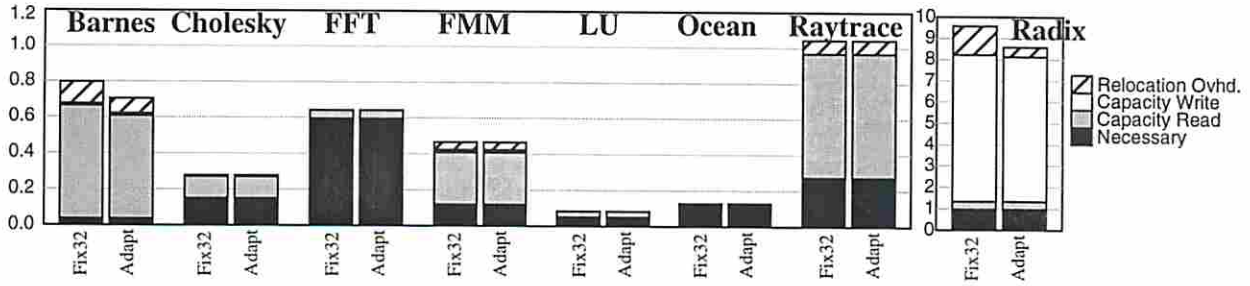

Figure 6. Cluster miss ratios (%) for the adaptive and fixed (32) relocation threshold policies for *ncp5*
The network cache in *ncp* maintains inclusion for dirty blocks only. The page cache size is 1/5 of total data set size. The adaptive threshold policy starts with a threshold of 32 and increases it if the page cache is thrashing.

In Figure 7, we display the node miss ratios to remote data for three systems with page caches. The first system has no NC. The second system, *ncp*, corresponds to R-NUMA [3]. The third system, *vbp*, integrates a page cache in a manner similar to R-NUMA, but uses a network victim cache. The measurements are made for four sizes of the page cache, expressed as a fraction of the application's entire data set size: 0, 1/9, 1/7, and 1/5. The last three points correspond to memory pressures of 90%, 87.5%, and 83.3%. In addition to the read and write miss ratios, we show the page relocation overhead at the top of each bar. This is obtained by scaling the page relocation ratio with the appropriate cost factor (225/30), to result in an equivalent amount of remote misses.

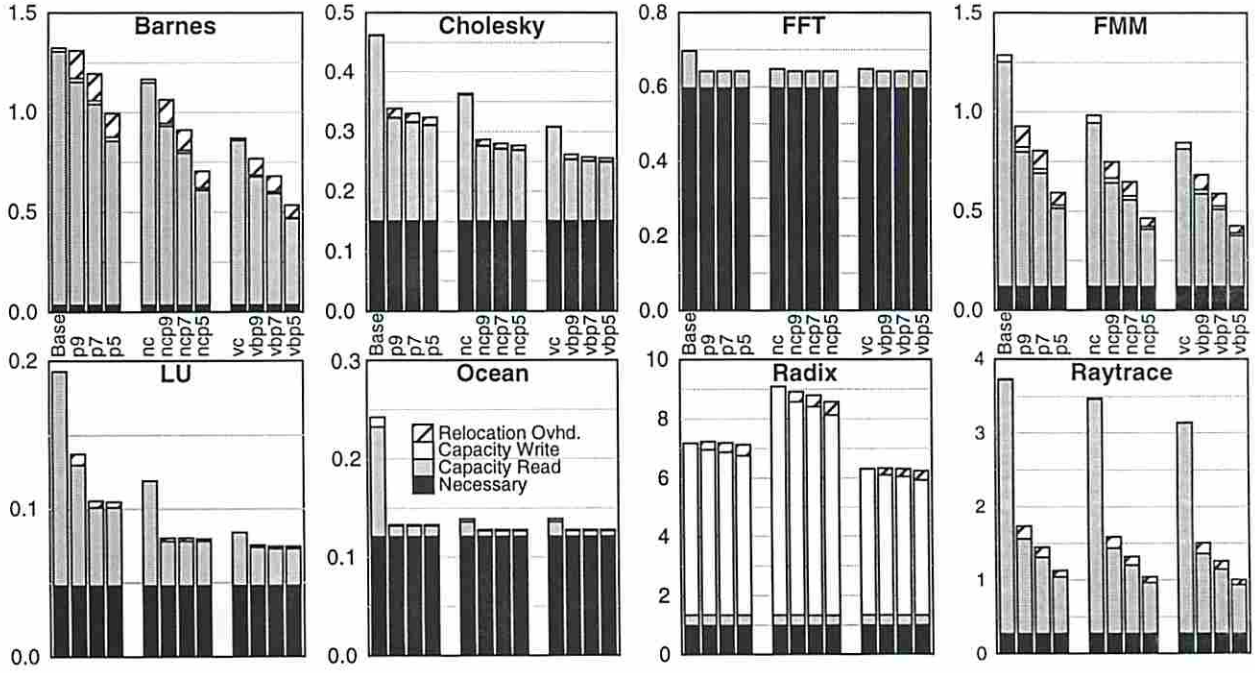

Figure 7. Comparison of cluster miss ratios (%) to remote data for several systems with page caches
Measurements are done without page cache and with three page cache sizes: 1/5, 1/7, and 1/9 of the data set size. The left four bars are for a system without NC. The middle four bars are for an NC that maintains inclusion for dirty blocks only. The right four bars are for a victim NC. The NC is 16KB, four-way set-assoc.

The 16KB NC clearly improves performance in both *ncp* and *vbp* over the system without NC. Although we do not present results here, our experiments with a 1KB network victim cache and a 1/7 page cache also show reductions as high as 14.7% (FMM) for the read miss ratio and 18.7% (Cholesky) for the relocation overhead. It is hard to attribute these improvements to the extra capacity offered by the victim

cache to the four 16KB processor caches (1.5%). Rather, the explanation is that NC is able to satisfy many conflict misses and to filter them out from the stream of directory requests that would increment the page relocation counters, thus reducing the relocation activity and the pollution of the page cache.

The better overall performance of the victim cache is clear even after a page cache is added. This is more evident when the page cache is small and the applications do not have the large spatial locality that prevents page cache fragmentation. There is no difference between *ncp* and *vbp* for FFT and Ocean. In both cases, the set of remote pages needing relocation is small and fits in the page cache without generating replacements. There is very little improvement for *vbp* in Cholesky because of the large spatial locality. Barnes, FMM, Radix, and Raytrace, applications with little spatial locality and irregular access patterns, benefit from the better hit ratio of the victim cache in *vbp*. When thrashing occurs, this better hit ratio helps prolong the lifetime of an entry in the page cache and increases its number of hits.

One can think of *vcp*'s better performance over *ncp* in terms of the amount of memory consumed by the page cache. The miss ratio of a page cache can be made as small as desired, provided enough memory. *vcp* needs less memory for the page cache than *ncp* in order to achieve a given performance level. This is an important advantage in systems where the page cache size can be adjusted dynamically based on multiple constraint factors, both in terms of resource utilization and frequency of adjustments.

6.2.1. Effects of the victim cache organization on systems with page caches

In Figure 8, we revisit the performance of the two ways of indexing the victim cache, but this time, in the presence of a page cache. When conflicts exist in the page-address indexed NC, it is expected that the page cache will help reduce them. Indeed, as compared with Figure 5, Cholesky shows less disparity between the miss ratios in the two cases, and Ocean and FFT show none. Overall, there is little difference between the two indexing methods, which indicates that a page-address indexed victim cache is feasible.

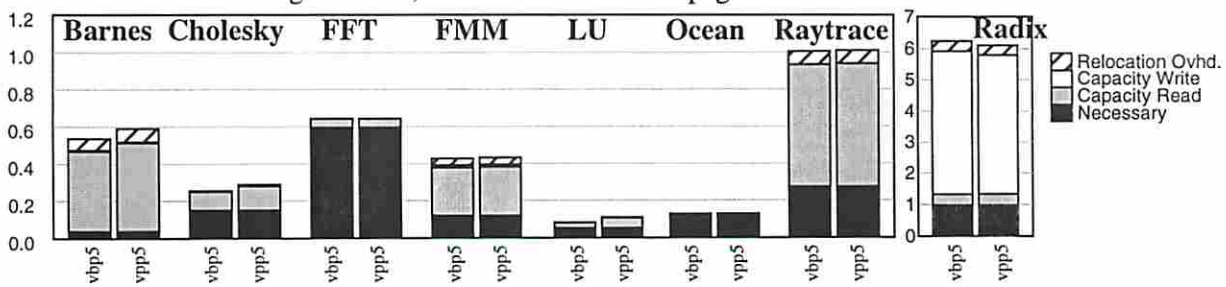

Figure 8. Cluster miss ratios (%) in systems with page cache for indexing network victim caches with block addresses and page addresses

The page cache size is 1/5 of the data set size. The victim NC is 16KB, four-way set-associative. *vbp* uses the LSB of the block address to compute the set in the NC; *vpp* uses the LSB of the page address.

6.3. The remote read stall

The remote read memory stall is the dominant component in the processor stalls under a release consistency memory model. Figure 9 shows the remote read stalls for several systems. The results are normalized with respect to the stalls in a system having an infinite but slow NC (DRAM). For reference, we indicate the stalls in a system with no NC (*base*) and in an ideal system with an infinite and fast SRAM NC (*NCS*). The *NCD* system has a 512KB DRAM NC. The systems with page caches *ncp*, *vbp*, and *vpp*, have a 512KB page cache and a 16KB NC. These three systems compare directly to *NCD*, because they use the same amount of DRAM. Because the applications are relatively small, a 512KB page cache is quite large for most of them and the differences between the three systems with page caches are less evident. Conse-

quently, we also show results for a page cache of size equal to 1/5 of the application's data set size.

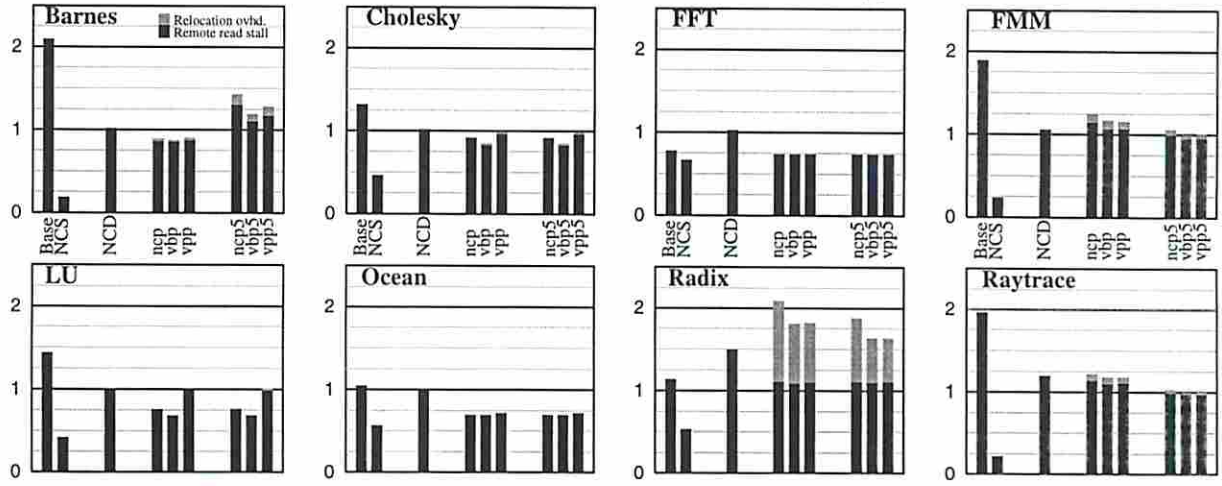

Figure 9. Remote read stalls

The remote read stalls are normalized with respect to a system with an infinite DRAM NC. Processor caches are 16KB, two-way set associative. The *base* system has no NC and no page cache. *NCS* has an infinite SRAM NC. *NCD* has a 512KB DRAM NC with full inclusion. *ncp*, *vbp*, and *vpp* have a 16KB four-way set-associative NC and a 512KB page cache. *ncp5*, *vbp5*, and *vpp5* have a page cache size in proportion to the application's data set size: 1/5 (for Cholesky, FMM, Ocean, and Raytrace this means more than 512KB). For systems with page caches we also indicate the overhead of page relocations at the top of each bar.

A first observation is that it is sometimes more desirable to have no NC at all than a slow one, even of infinite size. In FFT, there are significantly more necessary than capacity read misses and the penalty of accessing the slow NC cannot be offset. Ocean and Cholesky come close for the same reason.

The performance of systems with a 512 KB page cache, relative to *NCD*, falls into two categories, based on application characteristics. In applications with regular access patterns and high spatial locality (Cholesky, FFT, LU, and Ocean), the systems with page caches outperform *NCD*. The reason is that fragmentation in the page cache is low and the page cache miss ratio, hence the page relocation overheads, are small. On the other hand, when the application has irregular access patterns and little spatial locality (FMM, Radix, and Raytrace), *NCD* performs better. Fragmentation increases the overheads of maintaining the page cache significantly, although the actual remote stall is comparable to *NCD*. In Radix, an extreme case, the page and network caches are barely able to reduce the remote stall below the *base* configuration, but the page cache incurs quite an overhead for doing so. However, as will be seen later, the page cache has a positive effect on the data traffic, which is important in a high-traffic application. Finally, although Barnes falls in the same category as Radix, it has a smaller data set size and the 512KB page cache is able to hold much of the remote working set, in spite of fragmentation, thus performing better than *NCD*.

To compare the systems with page caches among themselves, we refer to the rightmost three bars, corresponding to a page cache size equal to one fifth of the data set size. In every application the victim cache organization shows improvement over *ncp*. The reason is improved effectiveness of the page cache. This improvement comes in two ways: less page relocation overhead and better cluster hit ratio. The page relocation overhead is reduced because the victim NC shields the directory (and the relocation counters) from the noise of conflict misses better than an NC with inclusion. The victim cache can also provide a surplus of capacity, albeit small. Overall, the relocation counters are less "eager" to trigger relocations. This extends the lifetime of an item in the page cache and the chances that it will amortize its relocation costs by satisfying more capacity misses. The better cluster hit ratio is due partly to the higher NC hit ratio and partly to less relocations. When pages are relocated in and out of the page cache, they can cause future misses because blocks must be evicted from the cluster due to the page re-mappings. Fewer relocations cause fewer evictions, hence a better cluster hit ratio.

The performance of *vpp* is very close to *vbp*. It is expected to be slightly worse in applications with large spatial locality (LU and Cholesky), but this is not always true. FFT shows some improvement for *vpp*, whereas Ocean shows no difference, mainly because all the pages with capacity misses are quickly migrated to a page cache that fits them. Afterwards, the conflicts in the NC sets stay low. The better performance of *vpp* is to be expected from the very start and throughout the execution of applications with irregular access patterns and sparse remote working sets, such as Radix. In this case, an NC indexed with page addresses can have a better hit ratio than an NC indexed by block addresses, especially when the NC works as a victim cache for the processor caches indexed by block addresses.

6.4. Data traffic

In Figure 10 we present the remote data traffic for the same systems as in the previous section. The data traffic includes read misses, write misses, and write-backs. As before, the results are normalized with respect to an infinite NC. Once again it becomes evident that systems with page caches perform just as well as *NCD* for applications with regular access patterns and large spatial locality, but have more trouble in dealing with the others. In the latter case, the victim cache is effective in reducing the traffic, especially in Radix, where the write and write-back traffic are significant. The improvement is less evident in Raytrace, where the read traffic dominates and is moderate in Barnes and FMM, where write traffic is low.

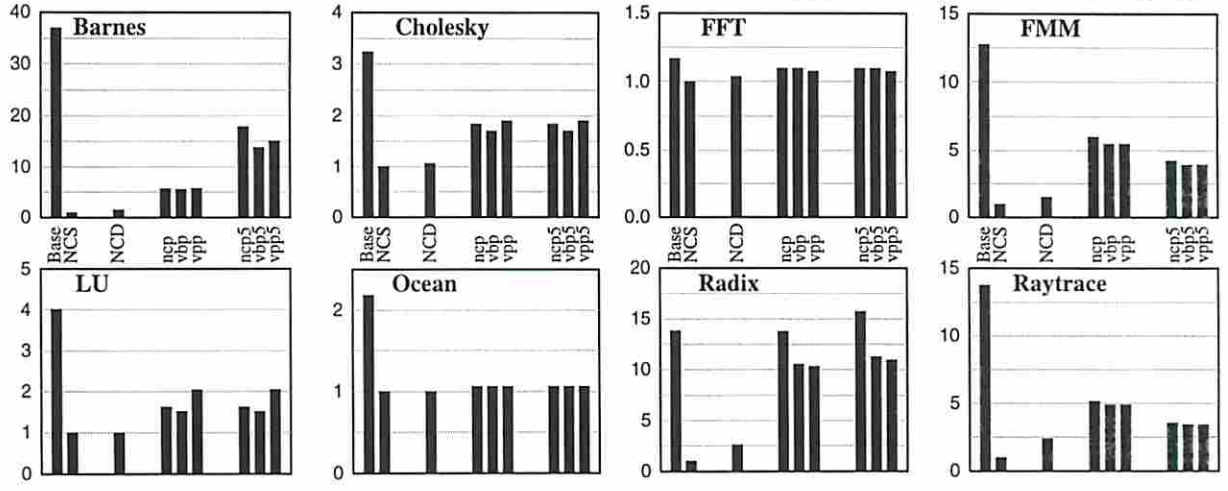

Figure 10. Remote data traffic

The results are normalized with respect to a system with an infinite NC. Processor caches are 16KB, two-way set associative. The *base* system has no NC and no page cache. *NCS* has an infinite SRAM NC. *NCD* has a 512KB DRAM NC with full inclusion. *ncp*, *vbp*, and *vpp* have a 16KB four-way set-associative NC and a 512KB page cache. *ncp5*, *vbp5*, and *vpp5* have a page cache size in proportion to the application's data set size: 1/5 (for Cholesky, FMM, Ocean, and Raytrace this means more than 512KB).

6.5. Performance of the victim cache with relocation counters

We compare the performance of the victim cache augmented with relocation counters (*vxp*) to R-NUMA (*ncp*) in Figure 11. The results are normalized with respect to an infinite DRAM NC. The page cache size is 1/5 and both systems use an adaptive relocation threshold. Because the victimization counters in *vxp* are incremented more often than *ncp*'s counters, we also present results for *vxp* when the initial value of the threshold is 64.

Even in applications with large spatial locality (Cholesky, Ocean) where conflicts in the victim cache and sharing of *vxp*'s counters are expected to be high, *vxp* performs as well as *ncp*. Likely, these two problems occur in *vxp* only in an initial phase of the execution; after much of the remote working set has

been relocated into the page cache, and the NC has a lighter load, these problems cease to exist. The reason LU performs worse in *vxp* is the same reason *vpp* performs worse than *ncp*. The small remote working set of LU would fit in the NC; page-address indexing, however, creates conflicts that relocate the working set into the page cache, where the access latency is larger than in the NC.

For the applications with small spatial locality (Barnes, FMM), *vxp* maintains the advantage of the victim cache over *ncp*. Radix has a slightly higher relocation overhead in *vxp,* simply because the number of victimizations is larger than the number of capacity misses and the page cache is completely inefficient in off-loading the NC. As can be seen, Radix shows much better improvement when the initial threshold is set to 64 than any other application. Even so, for Radix, *vxp* maintains the reduction of data traffic of systems with victim caches over *ncp*.

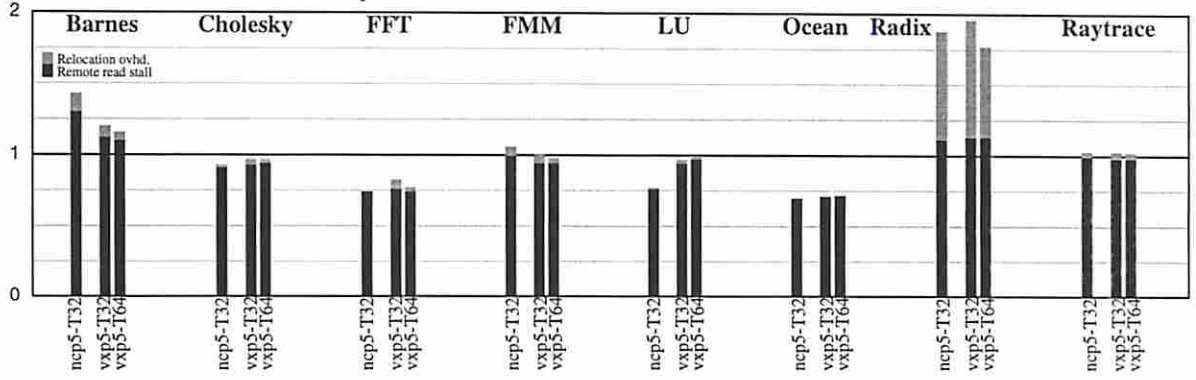

Figure 11. Remote read stalls in a systems with relocation counters controlled by the directory (*ncp5*) and by the victim cache (*vxp5*)

Results are normalized with respect to an infinite DRAM NC. The network caches are 16KB, four-way set-associative. In *ncp*, the NC maintains inclusion for dirty blocks. The NC in *vxp* is a victim cache. The page cache size is 1/5 of the data set size. *ncp* uses relocation counters based on the counts of capacity misses controlled by the directory; the threshold is adaptive and initially set to 32. *vxp* uses relocation counters based on counts of victimizations in the NC; the threshold is adaptive and initially set to 32 or 64.

7. Related Work

Victim caches were initially proposed for uniprocessors by Jouppi [7] as an effective method to deal with conflict misses in direct-mapped caches. Our proposed NC shares the same goal of hosting blocks victimized in the processor caches, but, in addition to dealing with conflict misses, it also handles remote capacity misses, some directly, and some by extension with a page cache. Researchers at Rice University [4] proposed the use of a victim cache for remote blocks, but as an additional resource between the processor caches and a large NC. In our proposal, the NC itself is a victim cache.

The benefits of relaxing inclusion between the processor caches and lower levels of the hierarchy have been recognized and exploited by other proposals. Non-inclusion between the NC and the caches for clean remote blocks was proposed by Fletcher et al. [4] and later used by R-NUMA [3]. Our method for integrating the network victim cache into the cluster bares resemblance to the two-level exclusive caching scheme [8] because the missed remote block is loaded directly into the first level cache. A similar technique was used in DYMA [10]. Essentially a COMA, DYMA uses the entire main memory as a victim cache. This, however, requires a fairly complex coherence protocol with 12 cache states and incurs longer memory access latencies. By comparison, our proposal only has five cache states and is a minor departure from a standard bus protocol. Our resulting bus protocol has some similarities with protocols used in bus-based COMAs [12], due to its replacement features for clean remote blocks.

Network (or cluster caches) can be found in many systems. DASH [15] has a 128 KB Remote Access Cache (RAC) which is mostly used to keep track of pending requests and support a dirty-shared

block state. Large DRAM NCs are used in the Convex Exemplar [1], as a configurable partition of main memory, and in the Sequent NUMA-Q [16], as a dedicated 32MB memory on the network interface card. S3.mp [18] also uses a DRAM NC of a configurable size. The performance of NUMA with large NCs has been compared to COMA by Zhang and Torrellas [25]. They found a better cost-effectiveness for systems with NCs for the SPLASH2 benchmarks.

The idea of page cache was first introduced in Simple COMA [21] as an extension of Virtual Shared Memory systems. One appeal of the page cache is that it can be hosted in main memory and it can be dynamically adjusted, as opposed to network caches which are only configurable at best. The other appeal, its simplicity coming from the software management of the page cache, is also its enemy, because of the overheads incurred. Two recent advances give new hopes for page caches. First, R-NUMA [3] is a proposal for unifying Simple COMA and CC-NUMA. R-NUMA uses fundamentally better mechanisms to trigger the page cache management software policies, which significantly cut the overhead. Second, the SGI Origin [14] is a proof that TLB shootdown, the highest overhead component of the page cache management, can be implemented more efficiently. The SGI Origin explores an alternative to large NCs by relying on aggressive page migration and replication, efficiently supported by the hardware. Although network caches do not fit the philosophy of the SGI Origin, it is possible that a small, very fast NC could shield the page migration and replication policies from the noise of conflict misses, thus improving system's performance.

8. Conclusions

In this paper we have evaluated design alternatives for remote data caches in clustered DSM systems and proposed new designs. Our results for the remote read stall and data traffic have shown that, with equal DRAM resources, systems with a network cache achieve better results than systems with a page cache mostly for applications with irregular access patterns, and large and sparse remote working sets. This, however, does not mean that systems with page caches are inferior in overall performance. They just need a bigger page cache to perform equally to a smaller NC. This could be a small price to pay considering the two payoffs. Firstly, the remote read stall **can** be reduced, by increasing the page cache size, below the levels attainable by an NC, no matter how big. Secondly, the page cache size can be adjusted dynamically, whereas the NC size is configurable at best. In multiprogrammed environments, with a mix of applications, this may be a decisive advantage.

Consequently, the challenge for systems with page caches is to make more efficient use of memory by reducing fragmentation, especially in applications with irregular access patterns, and large and sparse remote working sets. The improvements can come from both refining the mechanisms and the policies. We have shown the effectiveness of a network victim cache in reducing the page relocation rate and also the benefits of an adaptive threshold policy in eliminating page cache thrashing. A particular victim NC organization indexed with the page address unifies all the mechanisms that control page relocation, by attaching the relocation counters to the victim cache sets, rather than the directory entries. Clearly, there is still room for improvements to make the page cache more adaptive and to reduce the overheads of page relocation.

We believe that in future high-end CC-NUMA systems network victim caches combined with page caches will offer a real alternative to large and slow network caches, especially as the ratio between remote and local memory accesses settles around three or lower.

9. References

[1] G. Astfalk and T. Brewer. An Overview of the HP/Convex Exemplar Hardware. http://www.convex.com/tech_cache/ps/hw_ov.ps.

[2] J. Baer and W. Wang. On the Inclusion Property for Multi-Level Cache Hierarchies. In *Proceedings of the 15th Annual International Symposium on Computer Architecture*, pages 73-80. 1988.

[3] B. Falsafi and D.A. Wood. Reactive NUMA: A design for Unifying S-COMA and CC-NUMA. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, pages 229-240. June 1997.

[4] K.E. Fletcher, W.E. Speight, and J.K. Bennett. Techniques for Reducing the Impact of Inclusion in Shared Network Cache Multiprocessors. Rice ELEC TR 9413, Rice University. December 1994.

[5] E. Hagersten, A. Landin, and S. Haridi. DDM - A Cache-Only Memory Architecture. *IEEE Computer*, pages 44-54, September 1992.

[6] T. Joe. *COMA-F: A non-Hierarchical Cache Only Memory Architecture*. PhD thesis, Stanford University, 1995.

[7] N.P. Jouppi. Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pages 364-373. May 1990.

[8] N.P. Jouppi and S.J.E. Wilton. Tradeoffs in Two-Level On-Chip Caching. In *Proceedings of the 21st Annual International Symposium on Computer Architecture*, pages 34-45. 1994.

[9] Kendall Square Research. *KSR1 Technical Summary*. Waltham, MA. 1992.

[10] J. Kong and G. Lee. Relaxing the Inclusion Property in Cache Only Memory Architecture. In *Proceedings of EURO-PAR '96*. August 1996.

[11] M. Lam, E.E. Rothberg, and M.E. Wolf. The Cache Performance and Optimizations of Blocked Algorithms. In *Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 63-74. April 1991.

[12] A. Landin and F. Dahlgren. Bus-Based COMA - Reducing Traffic in Shared-Bus Multiprocessors. In *Proceedings of the 2nd International Symposium on High-Performance Computer Architecture*, pages 95-105. February 1996.

[13] R. LaRowe and C. Ellis. Experimental Comparison of Memory Management Policies for NUMA Multiprocessors. *ACM Transactions on Computer Systems*, 9(4), pages 319:363. November 1991.

[14] J. Laudon and D. Lenoski. The SGI Origin: A ccNUMA Highly Scalable Server. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, pages 241-251. June 1997.

[15] D. Lenoski et al. The Stanford DASH Multiprocessor. IEEE Computer, 25(3), pages 63-79, March 1992.

[16] T. Lovett and R. Clapp. STiNG: A CC-NUMA Computer System for the Commercial Marketplace. In *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, pages 308-317. May 1996.

[17] M. Marchetti, L. Kontothanassis, R. Bianchini, and M.L. Scott. Using Simple Page Placement Policies to Reduce the Cost of Cache Fills in Coherent Shared-Memory Systems. In *Proceedings of the 9th International Parallel Processing Symposium*. April 1995

[18] A. Nowatzyk et al. The S3.mp Scalable Shared Memory Multiprocessor. In *Proceedings of the 1995 International Conference on Parallel Processing*, pp. I-1-I-10, August 1995.

[19] M. Papamarcos and J. Patel. A Low Overhead Coherence Solution for Multiprocessors with Private

Cache Memories. In *Proceedings of the 11th Annual International Symposium on Computer Architecture*, pages 348-354, June 1984.

[20] S.K. Reinhardt, J.R. Larus, and D.A. Wood. Tempest and Typhoon: User-level Shared Memory. In *Proceedings of the 21st Annual International Symposium on Computer Architecture*, pages 325-337, April 1994.

[21] A. Saulsbury, T. Wilkinson, J. Carter, and A. Landin. An Argument for Simple COMA. In *Proceedings of the 1st International Symposium on High-Performance Computer Architecture,* pages 276-285. January 1995.

[22] C. Scheurich and M. Dubois. Dynamic Page Migration in Multiprocessors with Distributed Global Memory. In *IEEE Transactions on Computers*, 38(8), August 1989.

[23] B. Verghese, S. Devine, A. Gupta, and M. Rosenblum. Operating System Support for Improving Data Locality on CC-NUMA Compute Servers. In *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*. October 1996.

[24] S. Woo et al. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pages 24-36, June 1995.

[25] Z. Zhang and J. Torrellas. Reducing Remote Conflict Misses: NUMA with Remote Cache versus COMA. In *Proceedings of the 3rd International Symposium on High-Performance Computer Architecture*. January 1997.