Design Alternatives for Shared Memory Multiprocessors *

John B. Carter, Chen-Chi Kuo, Ravindra Kuramkote, Mark Swanson

{retrac, chenchi, kuramkot, swanson}@cs.utah.edu
WWW: http://www.cs.utah.edu/projects/avalanche

UUCS-98-011

Department of Computer Science
University of Utah, Salt Lake City, UT 84112

March 23, 1998

Abstract

In this paper, we consider the design alternatives available for building the next generation DSM machine (e.g., the choice of memory architecture, network technology, and amount and location of per-node remote data cache). To investigate this design space, we have simulated six applications on a wide variety of possible DSM architectures that employ significantly different caching techniques. We also examine the impact of using a special-purpose system interconnect designed specifically to support low latency DSM operation versus using a powerful off the shelf system interconnect. We have found that two architectures have the best combination of good average performance and reasonable worst case performance: CC-NUMA employing a moderate-sized DRAM remote access cache (RAC) and a hybrid CC-NUMA/S-COMA architecture called AS-COMA or *adaptable S-COMA*. Both pure CC-NUMA and pure S-COMA have serious performance problems for some applications, while CC-NUMA employing an SRAM RAC does not perform as well as the two architectures that employ larger DRAM caches. The paper concludes with several recommendations to designers of next-generation DSM machines, complete with a discussion of the issues that led to each recommendation so that designers can decide which ones are relevant to them given changes in technology and corporate priorities.

1 Introduction

Scalable hardware distributed shared memory (DSM) architectures have become increasingly popular for high-end compute servers. One of the purported advantages of shared memory multiprocessors compared to message passing multiprocessors is that they are easier to program, because

*This work was supported by the Space and Naval Warfare Systems Command (SPAWAR) and Advanced Research Projects Agency (ARPA), Communication and Memory Architectures for Scalable Parallel Computing, ARPA order #B990 under SPAWAR contract #N00039-95-C-0018

programmers are not forced to track the location of every piece of data that might be needed. However, naive exploitation of the shared memory abstraction can cause performance problems, because the performance of DSM multiprocessors is often limited by the amount of time spent waiting for remote memory accesses to be satisfied. When the overhead associated with accessing remote memory impacts performance, programmers are forced to spend significant effort managing data placement, migration, and replication – the very problems that shared memory is designed to hide from programmers. Thus, the value of DSM multiprocessor architectures is directly related to the extent to which observable remote memory latency can be reduced to an acceptable level.

The two basic approaches for addressing the memory latency problem are building latency-tolerating features into the microprocessor and reducing the average memory latency. Because of the growing gap between microprocessor cycle times and main memory latencies, modern microprocessors incorporate a variety of latency-tolerating features such as fine-grained multithreading, lockup free caches, split transaction memory busses, and out-of-order execution [1, 14, 19]. These features reduce the performance bottleneck of both local and remote memory latencies by allowing the processor to perform useful work while memory is being accessed. However, other than the fine-grained multithreading support of the Tera machine [1], which requires a large amount of parallelism and an expensive and proprietary microprocessor, these techniques can hide only a fraction of the total memory latency. Therefore, it is important to develop memory architectures that reduce the overhead of remote memory access.

Remote memory accesses fall into three different categories: (i) cold misses, (ii) coherent misses, and (iii) conflict/capacity misses, hereafter referred to simply as conflict misses. The frequency of cold and coherent misses depend on application access patterns, the coherency protocol used, and the initial memory allocation policy. In contrast, the frequency of conflict misses, a focus of this paper, depends on the amount of caching available for remote accesses. The remote memory overhead caused by conflict misses is governed by two issues: (i) the number of cycles required to satisfy each remote memory request and (ii) the frequency with which conflict misses to remote memory occur. The designers of high-end commercial DSM systems such as the SUN UE10000 [22], SGI Origin 2000 [9] and Mercury Interconnect Architecture [23] have put considerable effort into reducing the remote memory latency by developing specialized high speed interconnects. Pursuing an alternative architecture, the designers of STiNG [12] included a large DRAM network cache in the DSM controller to reduce the number of remote accesses. Simple-COMA (SCOMA) [21] proponents have espoused using part of the local DRAM memory as remote memory page cache. Recently researchers have suggested extending SCOMA to adapt to an hybrid architecture that combines the best properties of both the CC-NUMA and SCOMA memory models [5, 15, 8].

The designers of distributed shared memory systems face a plethora of design choices and accompanying open questions in balancing the cost of the system and its performance. If one wants to build a next generation scalable shared memory machine, what design should one choose? What are the design options? What are the cost/benefit ratios? Where are the sweet spots? Does adding a remote access cache (RAC) significantly help? If so, is it better to build a small but fast SRAM RAC or a larger but slower DRAM RAC? As an alternative to dedicating RAM to a RAC, one might consider using a portion of main memory as an additional local replication memory, by supporting an S-COMA or hybrid architecture. This last decision changes not only the cost factors, but also introduces additional operating system overhead. The utility of adding dedicated replication memory depends on the cost of remote memory accesses that are eliminated. This, in turn, introduces the question of interconnect price and complexity. Do any or all of these architectures reduce the frequency of remote accesses enough to allow the use of a less aggressive, and thus less costly, interconnect?

The goal of this paper is to attempt to answer these questions by analyzing the costs and benefits of the various methodologies on a variety of applications.

We considered five candidate architectures for next generation DSM machines: pure CC-NUMA [10, 9, 22], CC-NUMA extended to include either a DRAM remote access cache [12] (*remote access cache*) or an SRAM RAC [11] (SRAC), pure Simple COMA [21] (S-COMA), and a hybrid CC-NUMA/S-COMA architecture [5, 15, 8] we call AS-COMA [8] or *adaptable S-COMA*. Using detailed execution-driven simulation, we examined these five architectures using two interconnects of significantly differing performance characteristics on six applications. In our study, we found that two architectures have the best combination of good average performance and reasonable worst case performance: CC-NUMA employing a moderate-sized DRAM remote access cache (RAC) and a hybrid CC-NUMA/S-COMA architecture called AS-COMA or *adaptable S-COMA*. This result indicates that for the programs and network latencies that we considered, providing *large* remote data caches is more important than providing *fast* ones. We found that the performance of machines incorporating pure S-COMA, pure CC-NUMA, or CC-NUMA extended to include a small SRAM RAC lag noticeably behind the performance of the above two architectures

When deciding whether to build a CC-NUMA with a DRAC or an AS-COMA, the most important consideration is the memory access pattern of what the designer considers typical applications. If your typical applications have strong spatial locality and working set sizes that allow at least 10% of main memory to be used as a page cache, AS-COMA is the preferred option. If, however, your typical applications consume all of main memory or have poor spatial locality, CC-NUMA with a DRAC is the preferred option.

Finally, we found that the provision of a modest-sized DRAM RAC noticeably improves the performance of pure CC-NUMA machines, *even when the ratio of local to remote access latencies is as low as 1:3.* This result implies that the designers of the next generation SGI Origin 2000 should seriously consider adding a DRAC to their system, despite the excellent performance of their Spider interconnect.

The remainder of this paper is organized as follows. In Section 2 we describe the design of the different DSM architectures that we compared. We describe our simulation environment, test applications, and experiments in Section 3. We present the results of our detailed simulation experiments in Section 4, and compare our research with related work in Section 5. Finally, we draw conclusions and discuss possible future work in Section 6.

2 Design

In this section, we discuss organization of the DSM machines that we are going evaluate: CC-NUMA, CC-NUMA extended with RAC, S-COMA and AS-COMA.

2.1 Directory-based DSM Architectures

All the shared memory architectures that we consider share a common basic design, illustrated in Figure 1. Individual nodes are composed of a single commodity microprocessor with its own private processor caches connected to a coherent split-transaction memory bus. Also on the memory bus is a main memory controller with shared main memory and a distributed shared memory controller connected to a node interconnect. The aggregate main memory of the machine is distributed across all nodes. The processor, main memory controller, and DSM controller all snoop the coherent memory bus, looking for memory transactions to which they must respond.

The internals of the DSM controller are also shown in Figure 1. It consists of a memory bus snooper, a control unit that manages locally cached shared memory ("cache controller"), a control unit that retains state associated with shared memory whose "home" is the local main memory ("directory controller"), a network interface, and some local storage. In all the design alternatives that we explore the local storage contains DRAM which is used to store directory state. The shaded region which denotes RAC is present only in the two RAC configurations, while the page cache state region is only present in the SCOMA and AS-COMA models.

When a local processor makes an access to shared data that is not satisfied by its cache, a memory request is put on the coherent memory bus where it is observed by the DSM controller. The bus snooper detects that the request was made to shared memory and forward the request to the DSM cache controller. The DSM cache controller will then take one of the following three actions: (i) if the data is in main memory (home memory or page cache memory), a coherency response is given which allows the main memory controller to satisfy the request, (ii) if using a RAC model, a lookup is done in the cache in local storage and the memory request is satisfied on hit, (iii) Otherwise, the request is forwarded to the appropriate remote node. Once a response has been received, the DSM cache controller supplies the requested data to the processor, and potentially also stores it to main memory or RAC.

A remote request for data that is received across the interconnect is forwarded to the directory controller which tracks the status for each line of shared data for which it is the home node. If the remote request can be supplied using the contents of local memory, the directory controller simply responds with the requested data and updates its directory state. If the directory controller is unable to respond directly, e.g., because a remote node has a dirty copy of the requested cache line, it forwards the request to the appropriate node(s) and updates its directory state.

Examples of each of these architectures have been described elsewhere. This paper concentrates on comparing the various methodologies they use to reduce the remote access overhead due to conflict misses. This overhead can be represented as:

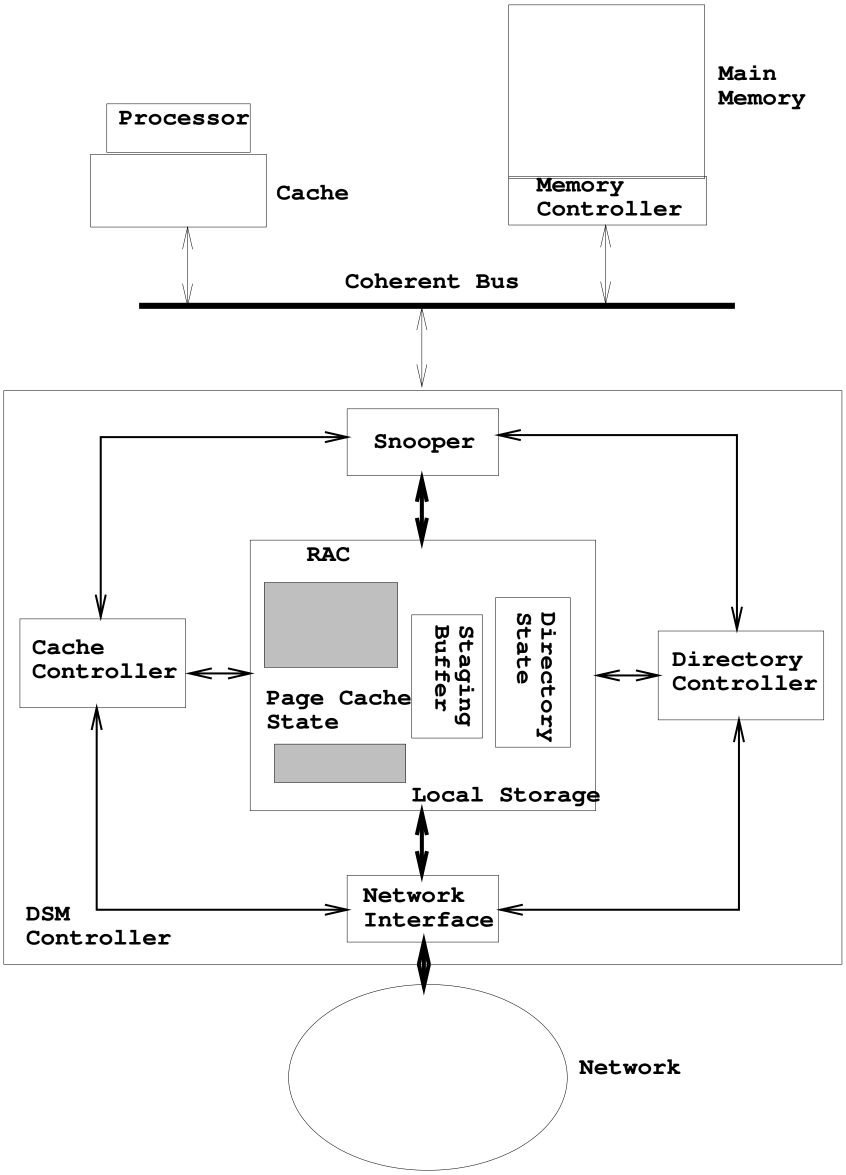


Figure 1 Typical Scalable Shared Memory Architecture

$(R_{pagecache} * L_{pagecache}) + (R_{srac} * L_{srac}) + (R_{drac} * L_{drac}) + (R_{rem} * L_{rem}) + KO.$

$R_{pagecache}$, R_{srac}, R_{drac} and R_{rem} represent the number of conflict misses that were satisfied by the page cache, SRAC, DRAC and remote memory. The $L_{pagecache}$, L_{srac}, L_{drac} and L_{rem} represent the latency while fetching the line from page cache, SRAC, DRAC and remote memory. KO represents the software overheads experienced by the S-COMA and AS-COMA models.

The Figure 2 summarizes the remote memory overhead and where one can invest to reduce it. The Figure 3 provides the cost in terms of the storage and complexity for each of the models. These will be explained in the following sections along with how each model works.

Model	Remote Overhead	Performance Factors
CC-NUMA	$(R_{rem} * L_{rem})$	Network speed
CC-NUMA (SRAC)	$(R_{srac} * L_{srac})+$ $(R_{rem} * L_{rem})$	1.Network speed 2. SRAM size and associativity
CC-NUMA (DRAC)	$(R_{srac} * L_{srac})+$ $(R_{rem} * L_{rem})$	1. Network speed 2. DRAM size and associativity
SCOMA	$(R_{pagecache} * L_{pagecache})+$ KO	1. Network speed 2. Software overhead
AS-COMA	$(R_{pagecache} * L_{pagecache})+$ $(R_{rem} * L_{rem}) + KO$	1. Network speed 2. Software overhead

Figure 2 Remote Memory Overhead of Various Models

Model	Storage Cost	Complexity
CC-NUMA	None	None
SRAC	SRAM	Controller for SRAM
DRAC	DRAM	Controller for DRAM
SCOMA	Page cache state: 1. 2 bits per block 2. 44 bits per page	1.Page cache state lookup 2. local $<->$ remote page map 3. Page-daemon and VM kernel
AS-COMA	Page cache state: 1. 2 bits per block 2. 44 bits per page Refetch Count: 6 bits per page per node	1.Page cache state controller 2. local $<->$ remote page map 3. Page-daemon and VM kernel 4. Refetch counter, comparator and interrupt generator

Figure 3 Cost and Complexity of Various Models

2.2 CC-NUMA

In CC-NUMA, a mapping from a global virtual address to the appropriate global physical address is created at the first page fault to that shared memory page. This mapping is inserted into the local page table and the TLB. If the home node of the page is not the local node, then the global physical address will contain that node number. Subsequently, when the local processor suffers a cache miss to a line in this shared data page, the DSM controller fetches a copy from the remote node, incurring a significant access delay.. Applications that suffer a large number of conflict misses to remote data perform poorly on CC-NUMAs [5]. Unfortunately, these applications are fairly common [17] because remotely homed data can be cached only in the relatively small processor cache.

The conflict miss cost in the CC-NUMA model is represented by $(R_{rem} * L_{rem})$, that is, all misses to shared memory with a remote home must be remote misses. To reduce this overhead, designers of such systems have to adopt a high speed interconnect to reduce (L_{rem}). Such an investment also reduces the cold and coherent access overhead, helping programs dominated by any of the three miss types.

2.3 CC-NUMA with RAC

In the RAC model, a non-inclusive[1] secondary cache for remote data is added to the DSM controller to help reduce conflict miss costs by reducing R_{rem}. The RAC model operates just as CC-NUMA except that a line that is brought from a remote node is also stored in the RAC. If the line is conflicted out of the processor cache and then re-referenced, it is supplied from the RAC if it is still present there. An SRAC is composed of SRAM which can provide short access times but will be relatively small due to the cost of SRAM. A DRAC is comprised of DRAM, and can thus be made quite large at reasonable cost, resulting in higher hit rates than a similarly costly SRAC. The DRAC hit rate will be offset by the longer access time of DRAM, however. Compared to CC-NUMA, these models entail additional cost for the SRAM or DRAM and for the cache controller to manage the RAC.

In the SRAC and DRAC models, the overhead is given by $(R_{srac} * L_{srac}) + (R_{rem} * L_{rem})$ and $(R_{drac} * L_{drac}) + (R_{rem} * L_{rem})$, respectively. In these models the remote overhead can be reduced either by increasing the RAC size, which in turn reduces R_{rem}, or by reducing L_{rem} or both. Whether the SRAC outperforms a DRAC depends on the SRAC and DRAC hit ratio and the SRAM to DRAM speed differential.

2.4 S-COMA

In the S-COMA model, the DSM controller and operating system cooperate to provide access to remotely homed data. In S-COMA, a mapping from a global virtual address to a local physical address is created at the first page fault to that shared memory page. The page fault handler selects an available page from the *page cache* in the *local* physical memory to use in the mapping. *Page cache state* in the DSM controller local storage that maps local physical pages to global physical pages is updated, as well as the set of *valid bits* for each S-COMA page, where each bit indicates whether a particular cache line in the page is valid. If there are no free S-COMA pages when a page fault occurs, the page fault handler selects an S-COMA page to replace and flushes the corresponding cache lines from the local processor cache prior to mapping the new S-COMA page.

When a local processor suffers a cache miss to remote data, the DSM cache controller examines the valid bit for the line. If the valid bit is set, the data can be supplied directly from main memory, thereby avoiding an expensive remote operation. If, however, the requested line is *invalid*, the DSM cache controller will perform a remote request to acquire a copy of the requested data. The returned line is written to the page cache and also supplied to the processor.

S-COMA's aggressive use of local memory to replicate remote shared data can significantly reduce R_{rem} when the *memory pressure* on a node is low. Memory pressure is the percentage of machine memory being used to store home pages or strictly local pages, and which are thus unavailable for use as S-COMA pages. For example, at 70% memory pressure, on average only

[1]Though non-inclusive, the RAC is not exclusive. Data may be in both the RAC and the processor cache at the same time. Conflict in the RAC do not cause invalidations in the processor cache.

30% of each node's pages are available for use in the page cache. Pure S-COMA's performance degrades rapidly for some applications as memory pressure increases. *All* remote data *must* be mapped to some local physical page before it can be accessed, so if the number of local physical pages available for S-COMA page replication is small, there is heavy contention for these pages. When the number of valid cache lines per S-COMA page is low, increasing memory pressure causes an S-COMA machine to thrash due to paging before a CC-NUMA machine would thrash due to cache misses. Given the high cost of page replacement, this can lead to dismal performance.

The S-COMA model requires DRAM in the DSM controller to store page state information [2]. Also, there is a slight increase in the complexity of the cache controller since it has to lookup the valid bit and also has to translate local page addresses to global page addresses and vice-versa. Finally, S-COMA imposes a software development cost in terms of modification to kernel VM system and page-out daemon software development.

In the SCOMA model, the conflict miss cost is represented by $(R_{pagecache} * L_{pagecache}) + (R_{rem} * L_{rem}) + KO$. Up to a certain application-dependent memory pressure threshold, page remapping does not occur and R_{rem} is zero. For example, if each node in the system has 100 pages and the application requires at each node 50 home pages and a maximum of 50 pages for replication, R_{rem} will be zero until 50% memory pressure. As the memory pressure increases beyond this threshold, R_{rem} increases as the pages in the page cache must be remapped, thus losing their effectiveness for satisfying conflict misses. Even worse, however, is that as memory pressure approaches 100%, page thrashing causes kernel overhead (KO) to become significant. This overhead includes: context switch time between application and page-out daemon, flushing of blocks from victim pages, page remapping, and additional misses that occur after the remapping.

2.5 AS-COMA

AS-COMA is a hybrid model that is similar to the S-COMA model. It differs from S-COMA by using the page cache only for *hot* remote pages. A page is considered *hot* if it is being accessed actively and lines within it suffer a lot of conflict misses. We use mechanisms similar to RNUMA [5] to identify *hot* pages. The directory controller maintains for each page a count of refetches from a node. When the count crosses a threshold, the directory controller informs of the *hot* page number by interrupting the node. Initially, AS-COMA handles the page faults identically to S-COMA. Once the number of pages in the page cache reaches a threshold where remapping will start to occur, the behavior of AS-COMA changes. In this phase, a pageout daemon runs periodically and goes through a victim eviction process wherein *cold* [3] pages in the page cache are selected for eviction. The valid blocks from each selected page are flushed from the processor cache and the page is added to the free page pool. The virtual page corresponding to this victim is then mapped

[2] 2 bits per line is needed to indicate the validity and the state of the line. Assuming a reasonable memory per node, 44 bits per page is needed to store local to remote and remote to local page translation.

[3] A page in the page cache not being actively used is termed a *cold* page. This can be determined by accumulating TLB reference bits.

to the global physical page back at the home node. Subsequent cache line misses to such pages are satisfied as in CC-NUMA. If enough free pages are available in the page cache, the pageout daemon remaps *hot* pages to one of the local page. Before doing that, the daemon will have to flush out all the blocks from the the processor cache.

By supporting both CC-NUMA and S-COMA access modes in the same machine, AS-COMA is able to exploit available local memory as a large RAC for CC-NUMA pages. By tracking refetch counts, it is able to select dynamically which CC-NUMA pages should populate the S-COMA cache based on access behavior.

AS-COMA entails all of the implementation costs of S-COMA as well as some additional costs. First, there is another slight increase in the complexity of the cache controller to maintain the refetch counts. Second, there is the requirement for storage to maintain the refetch count for each node and for each page. Finally, there is some additional software complexity in the page-out daemon to enable it to exploit the refetch counts in its remapping decisions.

AS-COMA's conflict miss cost is identical to that of the SCOMA model, up to the memory-pressure threshold at which page remapping begins in S-COMA. At this point, an effective AS-COMA will track close to (R_pagecache * L_pagecache), with only modest increases in R_{rem} up to some higher threshold, at which the page cache is no longer large enough to hold the hot pages. A perfect AS-COMA would simply degrade monotonically to the CC-NUMA cost, $R_{rem} * L_{rem}$), as a worst case at 100% memory pressure. Realizable AS-COMA models will fare worse than CC-NUMA at pressures somewhat less than 100%, due to the extra kernel overhead incurred before the system stabilizes.

AS-COMA differs from the other hybrid approaches in three ways: (i) it chooses cold pages for eviction from the DRAM page cache using local information; (ii) it uses S-COMA, rather than CC-NUMA, as the initial allocation policy when possible; and, (iii) it supports a graceful backoff algorithm to avoid thrashing when the number of free pages available in the memory becomes too small. This backoff algorithm is particularly important for avoiding excessive page thrashing and kernel overhead at high memory pressures [8].

3 Performance Evaluation

3.1 Experimental Setup

All experiments were performed using an execution-driven simulation of the XXX architecture[4]. Our simulation environment includes detailed simulation modules for a first level cache, system bus, memory controller, network interconnect, and DSM engine. It provides a multiprogrammed processor model with support for operating system code, so the effects of OS/user code interactions are modeled. The simulation environment includes a kernel based on 4.4BSD that provides scheduling, interrupt handling, memory management, and limited system call capabilities. The

[4]Architecture occluded to maintain anonymity.

modeled physical page size is 4KB. The VM system was modified to provide the page translation, allocation, and replacement support needed by the various distributed shared memory models. We extended the first touch algorithm [13] to equally distribute home pages to nodes by limiting the number of home pages that are allocated at each node using first touch. Once this limit is reached, remaining pages are allocated in a round robin fashion to nodes that have not reached the limit.

The modeled processor, DSM engine, and system bus are all clocked at 120MHZ. All cycle counts reported herein are with respect to this clock. The characteristics of the L1 cache, RACs, and network that we modeled are shown in Figure 4. In addition, we model a 4-bank main memory controller that can supply data from local memory in 58 cycles. The size of the main memory and the amount of free memory used for page caching was varied from application to application to test the different models under varying conditions. Given our SRAC and DRAC sizes, the ratio of SRAC to L1 cache size and DRAC to L1 cache size are 1:2 and 8:1, respectively, which we believe is reasonable for real machines.

We used a sequentially-consistent write-invalidate consistency protocol. DSM data is moved in 128-byte (4-line) chunks to amortize the cost of remote communication and reduce the memory overhead of DSM metadata. As part of a remote memory access, the DSM engine writes the received data back to the RAC or main memory as appropriate. Our CC-NUMA and AS-COMA models are not "pure," as we employ a 128-byte cache of the last remote data received as part of performing a 4-line fetch. This minor optimization had a larger impact on performance than we had anticipated, as is described in the next section.

We modeled two interconnects: a *fast* network where the remote to local memory access ratio was 3:1 and a *slow* network where the remote to local memory access ratio was 9:1. The fast network is intended to model a system with a system interconnect designed specifically to support low latency DSM operations, such as the Spider chip found in the SGI Origin 2000 [9]. The slow network is intended to model a system built using a powerful, but off the shelf, system interconnect such as Myrinet [2]. These interconnects represent two reasonable design alternatives that could be selected by a DSM system architect. Note that our network model only accounts for input contention.

Finally, Figure 5 shows the minimum latency required to satisfy a load or store from various locations in the global memory hierarchy. The average latency in our simulation is considerably

Component	Characteristics
L1 Cache	Size: 8-kilobytes. 32 byte lines, direct-mapped, virtually indexed, physically tagged, non-blocking, 1 up to one outstanding miss, 2 write buffers, 1-cycle hit latency
RACs	128 byte lines, direct-mapped, non-inclusive, non-blocking, up to 1 outstanding miss, Size/Latency: 4 kilobytes/23 cycles (SRAC) or 64 kilobytes/60 cycles (DRAC)
Networks	1 cycle propagation, 2X2 switch topology, port contention (only) modeled Fall through delay: 4 cycles (fast - 3:1) or 176 cycles (slow - 9:1)

Figure 4 Cache and Network Characteristics

higher than this minimum because of contention for various resources (bus, memory banks, networks, etc.) that we accurately model. simulation.

3.2 Benchmark Programs

We used six programs from the SPLASH-2 benchmark suite [24] in our study: `radix`, `fft`, `lu`, `barnes`, `cholesky`, and `ocean`. Figure 6 shows the inputs used for each test program. The column labeled *Home pages* indicates the number of shared data pages initially allocated at each node. These numbers indicate that each node manages from 0.5MB to 3MB of "home" data, with an average of 1.7MB per node over the six applications. We selected a processor cache size of 8KB, an SRAC size of 4K, and a DRAC size of 64K to keep the ratio between an average process's total working and the amount of caching it has available reasonable compared to real systems.

The *Maximum remote pages* column presented the maximum number of remote pages that each are accessed by a node for each application, which gives an indication of the size of the application's global working set. Finally, the *Ideal pressure* column is the memory pressure below which our S-COMA and AS-COMA machines act like a "perfect" S-COMA, meaning that every node has enough free memory to cache all remote pages that it will ever access. Below this memory pressure, S-COMA and AS-COMA never experience a conflict miss to remote data, nor will they suffer from kernel or page daemon overhead required to remap pages. Somewhat surprisingly, there is not a strong correlation between the ideal memory pressure for an application and how efficiently it executes on the various memory architectures. In particular, `radix` and `barnes` both accessed a large number of remote pages, yet `radix` performed quite poorly across the board while `barnes` performed quite well.

Due to their small default problem sizes and long execution times, `lu` and `fft` were run on just 4 nodes. All other applications were run on 8 nodes.

Finally, Figure 7 shows the amount of memory required to store S-COMA or AS-COMAs metadata. S-COMA requires 108 bits per page, while AS-COMA requires 132 bits per page for a 4 node architecture or 156 bits per page for 8 node architecture (from Figure 3). The *minimum* overhead represents the amount of metadata needed to manage 10% of main memory as a page cache (i.e., at 90% memory pressure), while the *maximum* overhead represents the amount of

Data Location	Latency
L1 Cache	1 cycle
Local Memory	58 cycles
SRAC	23 cycles
DRAC	60 cycles
Remote Memory (fast network)	147 cycles
Remote Memory (slow network)	491 cycles

Figure 5 Minimum Access Latency

metadata needed at the "ideal" memory pressure where all remote pages ever accessed by a node can be cached locally. The amount of storage required to store S-COMA/AS-COMA metadata is an important consideration, since this storage should be significantly smaller than a typical DRAC size for S-COMA/AS-COMA to make sense economically. Conveniently, it is.

4 Results

Figures 8 and 9 present the relative execution time of our six applications for each of the five memory models using both a slow and a fast interconnect. In addition to raw performance, we present a breakdown of where each program spent its time: performing *user-level* operations, stalled on shared memory *shmem*, or performing *kernel* operations, such as page replacement or process synchronization. All results reported below are for the parallel phase of the applications.

The bars in Figures 8 and 9 represent pure CC-NUMA, pure S-COMA, CC-NUMA augmented with an 8-kilobyte SRAM RAC (SRAC), CC-NUMA augmented by 64-kilobyte DRAM RAC (DRAC), and a hybrid CC-NUMA/S-COMA architecture (AS-COMA). Each architecture has two bars, one for each network. For S-COMA and AS-COMA, we simulated a number of memory pressures between 10% and 90%. This lets us see how well they perform when a large number of main memory pages are available for caching remote data (low memory pressure), and how stable

Program	Input parameters	Home Pages (per node)	Maximum Remote Pages	Ideal Pressure
`radix`	1M Keys, Radix = 1024	259	1306	17
`FFT`	256K Points, tuned for cache sizes	781	990	44
`LU`	1024x1024 matrix, 16x16 blocks, contiguous	514	405	56
`barnes`	16K particles	102	552	16
`cholesky`	`tk16` input	491	902	36
`ocean`	258x258 ocean	473	356	57

Figure 6 Programs and Problem Sizes Used in Experiments

Program	S-COMA	AS-COMA
`radix`	.38 - 17	.55 - 24
`fft`	1.1 - 13	1.4 - 15
`lu`	.76 - 5	.93 - 6
`barnes`	.15 - 7	.22 - 10
`cholesky`	.72 - 11	1.04 - 17
`ocean`	.69 - 4	1 - 6

Figure 7 Minimum and Maximum Storage Requirements for S-COMA and AS-COMA Metadata (kilobytes)

they are when the page cache shrinks to almost nothing (high memory pressure). All results are scaled relative to the performance of a "pure" CC-NUMA machine with a fast network, a model similar to the SGI Origin 2000 [9].

Figures 10 and 11 illustrates the effectiveness of the remote data caches employed by the different architectures by showing where cache misses to remote shared data are satisfied. An *S-COMA* miss is satisfied from the *local* page cache. *RAC* misses are satisfied from the *local* RAC (SRAM or DRAM). *COLD* misses are necessarily satisfied on a *remote* home node. Finally, *C/C* misses represent conflict/capacity misses that are not satisfied by a local RAC or S-COMA page, and thus result in *remote* accesses.

Looking at Figures 8 and 9, we can divide the six applications into roughly three groups: (i) applications for which the DSM memory architecture mattered very little (`ocean` and `fft`), (ii) applications that access moderate to high amounts of remote data and exhibit good spatial locality in these accesses (`barnes`, `lu`, and `cholesky`), and (iii) applications that access moderate to high amounts of remote data, but exhibit poor spatial locality in these accesses (`radix`). We consider each category in turn.

Neither `ocean` nor `fft` suffer a significant number of conflict or capacity misses to remote data. Although the results shown in Figure 11 make it appear that `ocean` suffers a high number of capacity misses for the CC-NUMA architectures, it turns out that using a simple first touch page allocation policy results in almost perfect page placement. Only 2% of `ocean`'s cache misses are to remote data, so the choice of DSM architecture or interconnect latency is largely irrelevant. Similarly, `fft`'s remote data set fits almost entirely in an 8-kilobyte L1 cache, so after the initial cold misses needed to load the relevant portions of remote memory into cache, `fft` suffers very few capacity or conflict misses to remote data. Those few misses are almost entirely satisfied by even the 128-byte RAC used in the "pure" CC-NUMA architecture, so once again, the choice of memory architecture is largely moot. In the case of `fft`, however, a faster network reduces execution time by approximately 5%. For both `ocean` and `fft`, the kernel overhead required to swap pages in S-COMA increases execution time by up to 40% at high memory pressures.

The architectures that are able to cache significant amounts of remote data significantly outperform "pure" CC-NUMA on the problems that access moderate to high amounts of remote data with good spatial locality (`barnes`, `lu`, and `cholesky`). `barnes`, in particular, exhibits very high spatial locality – it tends to access large dense regions of remote memory that can make good use of S-COMA pages. For this reason, AS-COMA executes `barnes` approximately twice as fast as pure CC-NUMA and 15% faster than a CC-NUMA with DRAC when the systems are connected via a slow interconnect. The reason for this is clear from Figure 10 - AS-COMA is able to convert the relatively high number of conflict misses in all of the CC-NUMA variants into local S-COMA hits. For `lu`, AS-COMA demonstrates a similar, but less dramatic, performance advantage over the

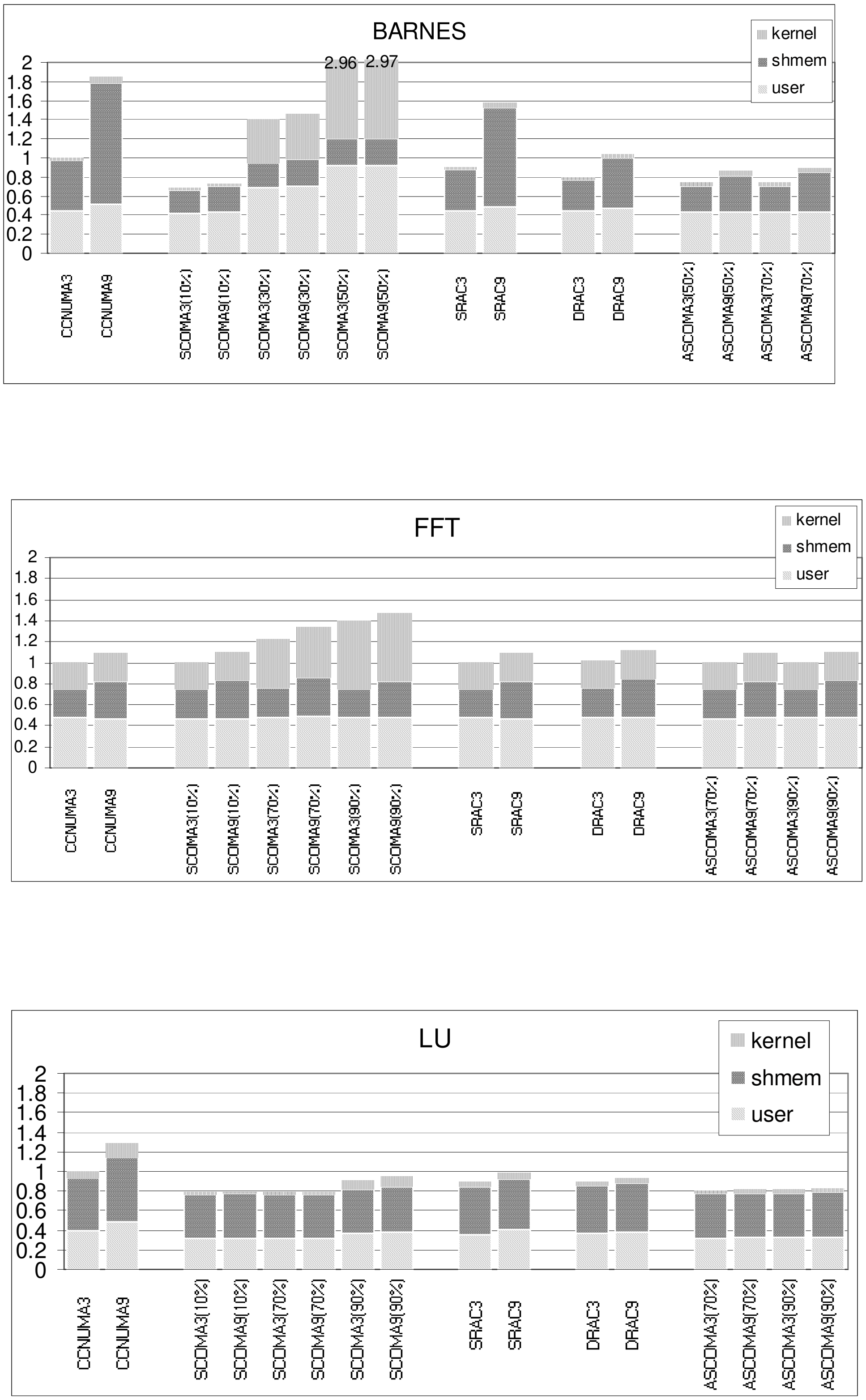
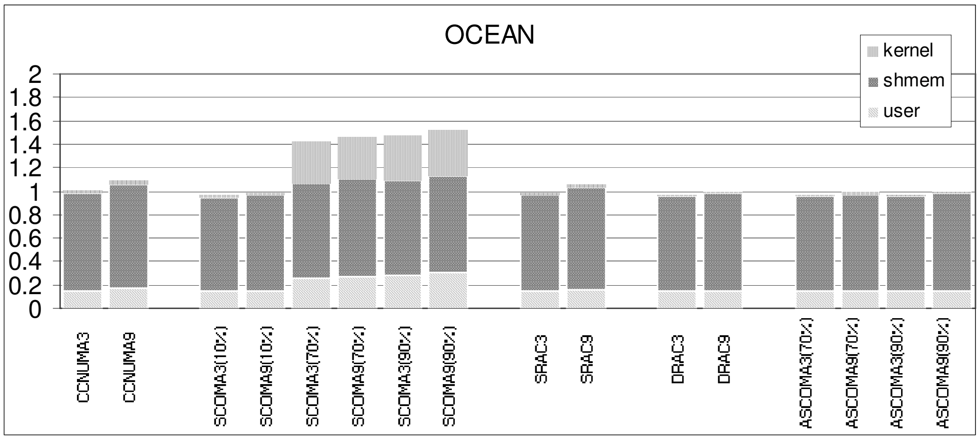
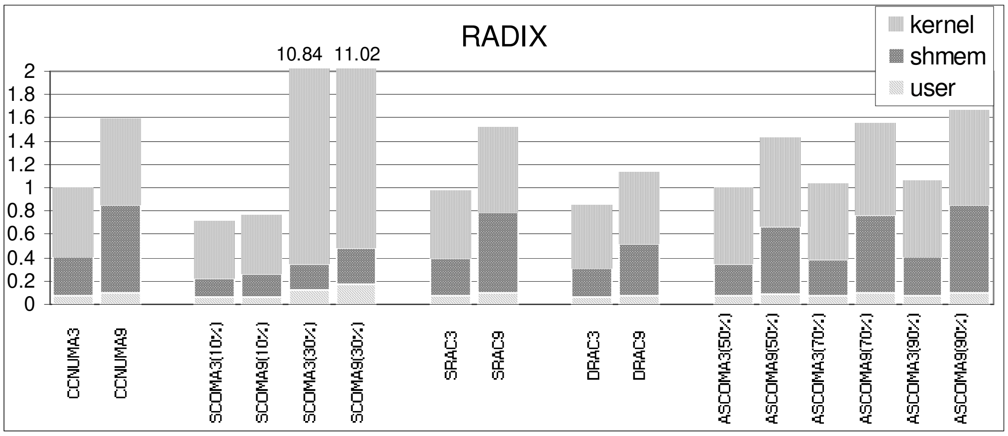


Figure 8 Relative Execution Times for `barnes`, `fft` and `lu`





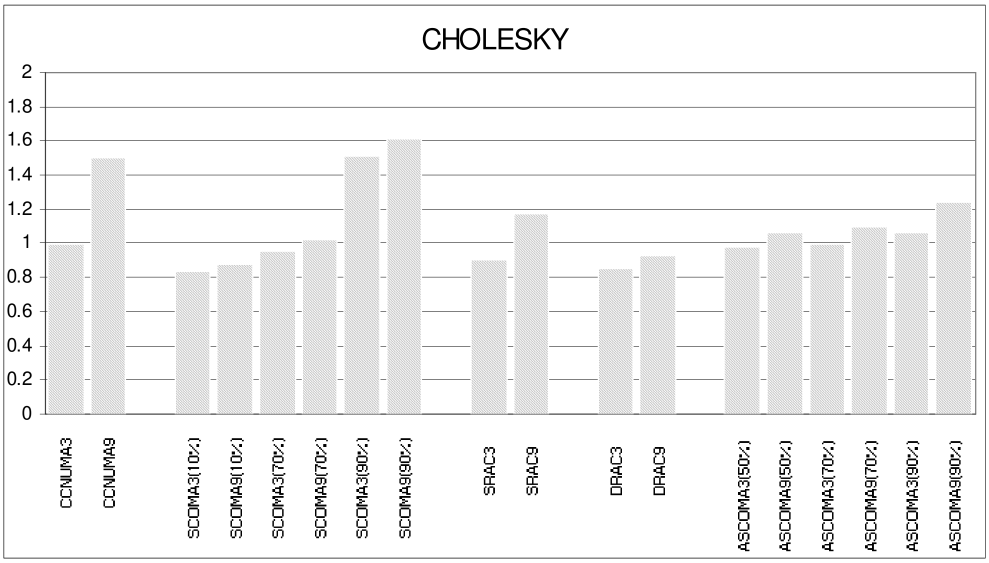
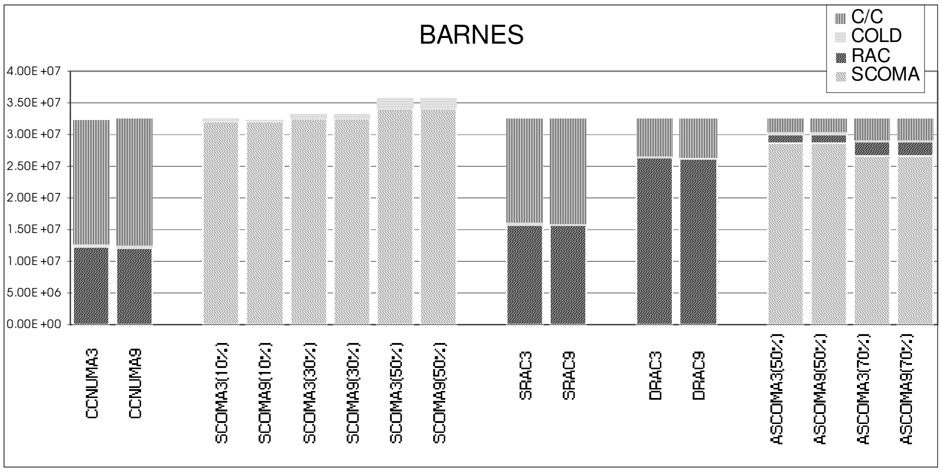
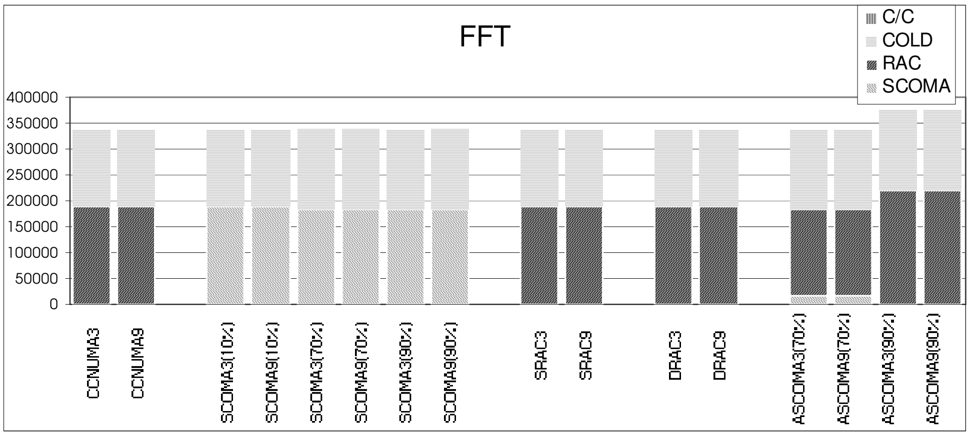


Figure 9 Relative Execution Times for `ocean`, `radix`, and `cholesky`





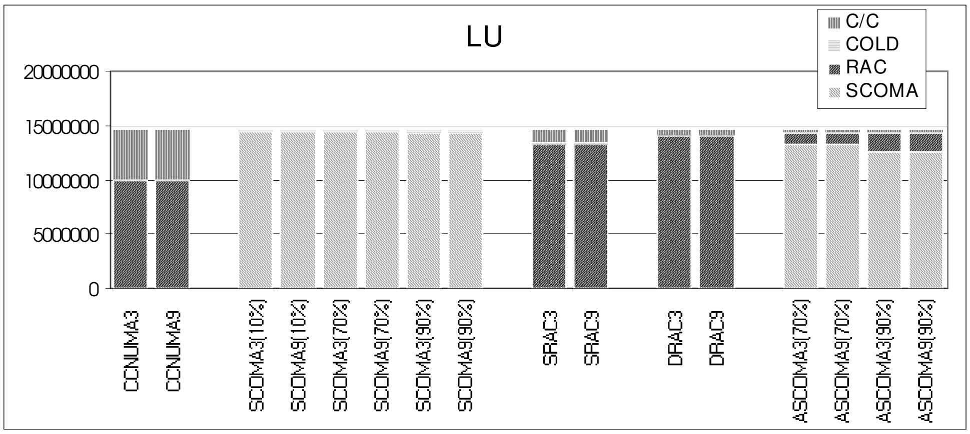


Figure 10 Where Cache Misses Were Satisfied for `barnes`, `fft` and `lu`

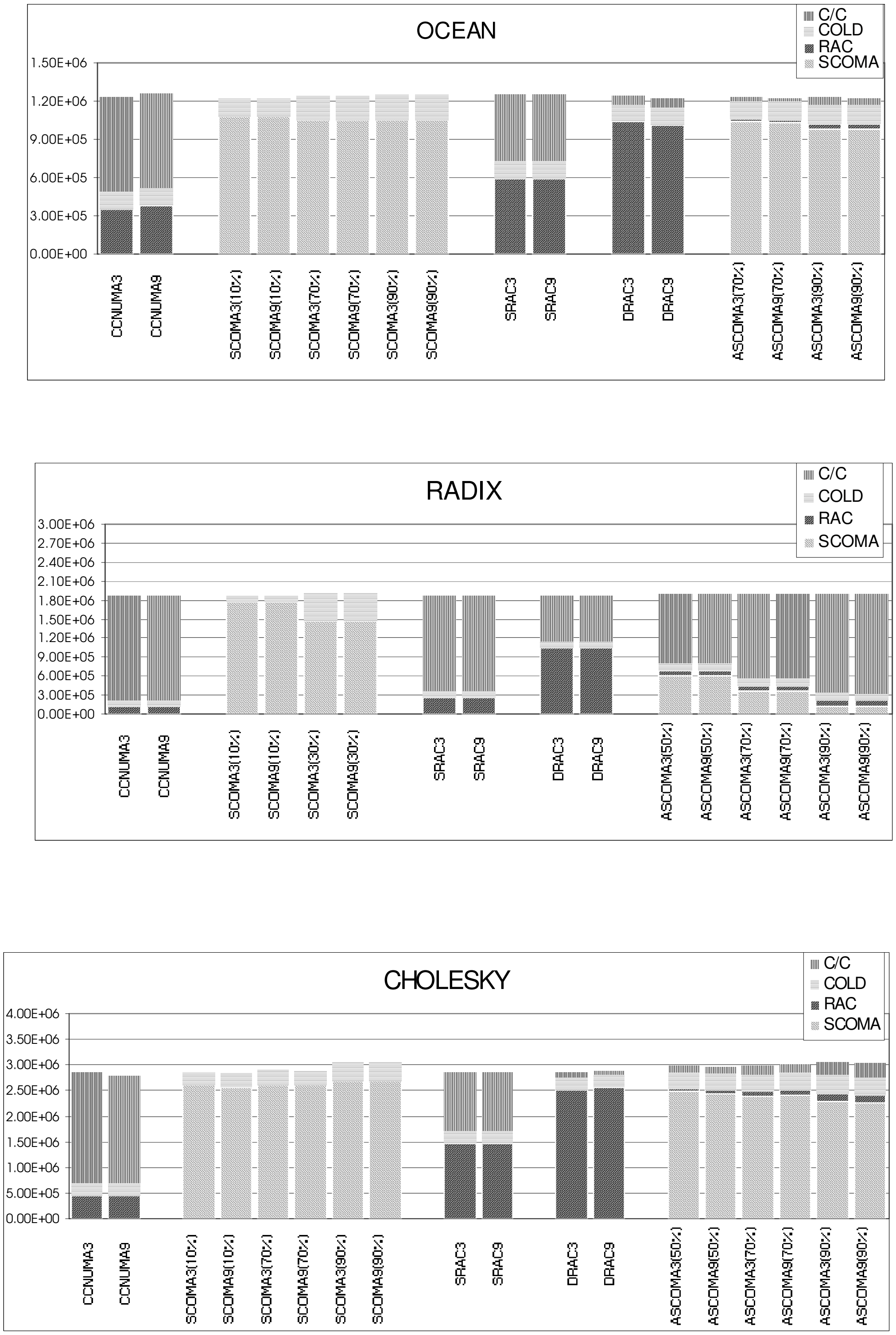


Figure 11 Where Cache Misses Were Satisfied for `ocean`, `radix`, and `cholesky`

CC-NUMA variants for the same reason. Finally, `cholesky` is primarily synchronization-bound, and thus not dramatically impacted by the choice of DSM architecture[5].

With a low latency interconnect, all of the architectures perform approximately equally, although CC-NUMA with a DRAC performs slightly better than the alternatives other than S-COMA at very memory pressures. With a high latency interconnect, the DRAC-based architecture completes `cholesky` in under two-thirds the time of the pure CC-NUMA machine, and 5-20% faster than the AS-COMA machine, depending on the memory pressure.

Finally, `radix` is a DSM architect's nightmare - it exhibits almost no spatial locality, as every processor accesses pseudo-random portions of every page of shared data during every iteration of the sort. This effect can be seen in the high remote miss rate experienced by every DSM architecture. This phenomenon causes pure S-COMA's performance to tail off dramatically due to thrashing once memory pressures exceed 17% (see Table 6). Even though `radix` accessed more remote pages than any other program, as illustrated in Figure 6, the 64-kilobyte DRAC architecture outperformed AS-COMA, with its larger but more coarsely allocated DRAM cache. This occurs because `radix`'s locality is so poor that the AS-COMA page cache is very poorly utilized - only a small number of cache lines tend to be active in any given S-COMA page at a time. Thus, in the case of `radix`, and similar applications, a large cache managed at a fine granularity is important.

We draw the following conclusions from the data presented in Figures 8 through 11:

- Overall, the CC-NUMA with DRAC and AS-COMA architectures have the best combination of good average performance and reasonable worst case performance. This indicates that for the programs and network latencies that we considered, providing *large* remote data caches is more important than providing *fast* ones.

- If your typical applications have strong spatial locality and working set sizes that allow at least 10% of main memory to be used as a page cache, AS-COMA is the preferred option. If, however, your typical applications consume all of main memory or have poor spatial locality, CC-NUMA with a modest-sized DRAC is the preferred option.

- Pure S-COMA suffers serious performance problems in medium-to-high memory pressure situations for applications with poor spatial locality, `radix` being the extreme example. In fact, in these circumstances, pure S-COMA's performance is so poor that we recommend that architects interested in providing S-COMA-like page caching seriously consider using a hybrid solution. AS-COMA's performance was never more than 5% worse than the equivalent pure CC-NUMA machine, even for `radix`.

- For pure CC-NUMA, providing a fast network improves performance by as much as 80% (e.g.,`barnes`), but often has negligible impact on performance (e.g., `fft` and `ocean`). Even

[5] We inadvertently collected the breakdown of where `cholesky` spends its time on the complete run, rather than just the parallel phase. Unfortunately, we discovered this error too late to re-run the necessary simulations in time to include their results here. Upon acceptance, these numbers will appear in the final paper.

with a fast network, the addition of a modest-sized DRAC can improve performance of a CC-NUMA machine 5-50% (e.g., `barnes`, `cholesky`, `lu`, and `radix`). This result implies that the designers of the next generation SGI Origin 2000 should seriously consider adding a DRAC to their system, despite the excellent performance of their Spider interconnect.

- If faced with the decision between spending engineering resources on the development of an extremely low latency network or a complex and powerful DSM controller, it is interesting to note that AS-COMA performed almost as well on average with a slow (9:1) network as pure CC-NUMA did with a fast (3:1) network. When coupled with a modest-sized DRAC, however, CC-NUMA with a fast network was clearly superior to an AS-COMA with a slow network. Finally, we note that fetching 128 bytes per remote cache fill, rather than the minimum 32 bytes required to satisfy a cache miss, removed a surprisingly high number of remote operations in both the "pure" CC-NUMA and AS-COMA models. This effect can be seen via the high percentage of remote misses satisfied in the 128-byte RAC employed in these systems. Upon further investigation, we determined that the effect is caused by a combination of factors. First, we are essentially prefetching three extra cache lines for sequentially accessed data. Second, we observed frequent conflicts between local and shared data caused by the small size of the L1 cache (8 kilobytes), and the 128-byte RAC allowed a large percentage of the resulting conflict misses to be satisfied locally.

5 Related Work

There are a number of past as well as ongoing efforts in the area of directory based hardware DSM architectures. This section details these research efforts. However, we limit our discussion to systems that used commodity processors and commodity coherent busses.

The Stanford DASH multiprocessor [11, 10], was one of the first systems [6] to use a directory-based cache design. DASH supported a CC-NUMA model and had 128KB of SRAM RAC in its DSM controller which was able to reduce remote traffic by 8-23%. The DASH system had a local access latency of 22 cycles and a remote access latency of 61 cycles, giving a remote to local access ratio of 3. The Origin 2000 system [9] is similar to the DASH system in many aspects except that it does not have a RAC, and it uses an extended coherence protocol which is robust to deadlock and out-of-order delivery of messages. The Origin system uses a very high speed interconnect based on SGI Spider router chip [6]. The Origin 2000 has a local memory access time of 310ns and a remote memory access time of 773ns for an 8 node system. The remote to local access ratio is approximately 2.5. The use of the high speed interconnect for distributed I/O justifies some of its cost. To alleviate the capacity/conflict misses, the system has support in hardware to aid in page migration. However, we did not consider its effects in our study since page migration to date has only been successful for read-only or non-shared data, which significantly limits their effectiveness.

[6] The MIT Alewife machine [3, 4] was the other contemporary machine to implement directory based shared memory.

The STiNG CC-NUMA machine [12] uses an SCI-based coherent interconnect. The system has a 4-way associative 32MB DRAM RAC. The average local access time ranged from .2 to .3usec and the remote time ranged from 3 to 10usec. Though the system has some cost-effective features such as cheap network and DRAC, the use of SCI coherent interface and occupancy in the controller, limit its performance [12].

Simple-COMA (S-COMA) systems [21, 18] combine the best of the DVM and the COMA systems [7] to use DRAM for remote memory replication. When a node accesses a page, an S-COMA machine allocates pages in local physical memory for replication, as is done by a DVM system. However, instead of fetching the whole page, the S-COMA DSM controller fetches each individual block from the home node on a cache miss. The Tempest and Typhoon systems [18] used a page cache known as *stache* to support S-COMA and fine grain software access control to detect and fetch invalid blocks in the page as they were accessed.

The S3.mp multiprocessor system [16] was developed with a goal of using a hardware supported DSM system in a spatially distributed system connected by a local area network. For the interconnect it used a new CMOS serial link that supported greater than 1Gbit/sec transfer rate. The S3.mp system was one of the first systems to support both CC-NUMA and S-COMA models. However, it did not demonstrate the hardware and software support necessary for a hybrid model. The *Reactive NUMA* [5] system is similar to the hybrid model we detailed in this study. The R-NUMA study showed the usefulness of a hybrid model by showing the improvement in performance over CC-NUMA and S-COMA for most of the applications with the worst performance penalty being 56%. However, the study did not include the performance of the hybrid model under different memory pressures, especially high pressures. Because of this the study did not research the back-off techniques necessary under high memory pressure. The victim-cache-NUMA (VC-NUMA) system [15] showed the problems of hybrid models at high memory pressure and suggested using a victim cache and reducing the number of *refetch counts*. However, their victim cache solution requires modifications to the processor cache controller and changes in the bus protocol. Their study had some similarities to our study as they compared against SRAC and DRAC. However, the VC-NUMA study did not explore varying the network latency.

6 Conclusions

In this paper, we have carefully studied the design alternatives available for building the next generation of scalable shared memory multiprocessors. From a candidate field of five major DSM architectures that includes all of the major DSM alternatives currently being proposed (three varieties of CC-NUMA, S-COMA, and a hybrid CC-NUMA/S-COMA architecture), we have identified two that appear to hold the most promise: CC-NUMA enhanced with a large DRAM remote access cache and AS-COMA, a hybrid CC-NUMA/S-COMA architecture.

Extending conventional CC-NUMA designs to include a large DRAM RAC, as is done in the Sequent STiNG [12], and employing a fast special-purpose network, as is done in the SGI Origin 2000 [9], would be the conservative approach for extending DSM architectures to the next generation. However, there are strong indications from this work and that of others [5, 15] that a hybrid CC-NUMA/S-COMA architecture such as AS-COMA holds at least as much promise as CC-NUMA with a large DRAC. AS-COMA, and the other hybrid CC-NUMA/S-COMA architectures, addresses S-COMA's primary failing: its instability under high memory pressures. The hybrid architectures all benefit from S-COMA's primary strength: the ability to cache remote data in generic system DRAM. Since AS-COMA caches data in main memory, it is easy to increase the size of an AS-COMA page cache - the need for high-speed tags and the restricted use of CC-NUMA RAC memory makes it less likely that this memory can be extended easily. In addition, memory added to an AS-COMA page cache can be used effectively by non-DSM applications, since it is, after all, just normal system memory.

In addition, we have identified the value of adding a DRAC even to CC-NUMA machines with very low latency networks, demonstrated the importance of considering hybrid CC-NUMA/S-COMA architectures to address S-COMA's inability to handle high memory pressure gracefully, and suggested a number of ways that DSM designers can tune their architectures.

To continue this work, we plan to explore additional design parameters that should be considered for the next generation of DSM architectures, such as set-associative RACs and putting processors on memory chips [20]. We also plan to continue to investigate ways to reduce the system software overhead associated with S-COMA architecture, as this software overhead seems to be the primary performance limiting factor for these architectures. Finally, we intend to extend both our simulation environment and set of applications so that we can evaluate a wider variety of design alternatives used in a larger number of ways.

References

[1] R. Alverson, D. Callahan, D. Cummings, B. Koblenz, A. Porterfield, and B. Smith. The Tera computer system. In *Proceedings of the 1990 International Conference on Supercomputing*, pages 1–6, September 1990.

[2] N.J. Boden, D. Cohen, R.E. Felderman, A.E. Kulawik, C.L. Seitz, J.N. Seizovic, and W.-K. Su. Myrinet – A gigabit-per-second local-area network. *IEEE MICRO*, 15(1):29–36, February 1995.

[3] D. Chaiken and A. Agarwal. Software-extended coherent shared memory: Performance and cost. In *Proceedings of the 21st Annual International Symposium on Computer Architecture*, pages 314–324, April 1994.

[4] D. Chaiken, J. Kubiatowicz, and A. Agarwal. LimitLESS directories: A scalable cache coherence scheme. In *Proceedings of the 4th Symposium on Architectural Support for Programming Languages and Operating Systems*, pages 224–234, April 1991.

[5] 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.

[6] M. Galles. Scalable pipelined interconnect for distributed endpoint routing. In *In Hot Interconnects'96*, 1996.

[7] E. Hagersten, A. Landin, and S. Haridi. DDM – A cache-only memory architecture. *IEEE Computer*, 25(9):241–248, September 1992.

[8] Chen-Chi Kuo, J. Carter, R. Kuramkote, and M. Swanson. As-coma: An adaptive hybrid shared memory architecture. Technical report, University of Utah - Computer Science Department, March 1998.

[9] J. Laudon and D. Lenoski. The SGI Origin: A ccNUMA highly scalable server. In *SIGARCH97*, pages 241–251, June 1997.

[10] D. Lenoski, J. Laudon, K. Gharachorloo, A. Gupta, and J. Hennessy. The directory-based cache coherence protocol for the DASH multiprocessor. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pages 148–159, May 1990.

[11] D. Lenoski, J. Laudon, K. Gharachorloo, W.-D. Weber, A. Gupta, J. Hennessy, M. Horowitz, and M. S. Lam. The Stanford DASH multiprocessor. *IEEE Computer*, 25(3):63–79, March 1992.

[12] T. Lovett and R. Clapp. STiNG: A CC-NUMA compute system for the commercial marketplace. In *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, pages 308–317, May 1996.

[13] M. Marchetti, L. Kontothonassis, R. Bianchini, and M.L. Scott. Using simple page placement policies to reduce the code of cache fills in coherent shared-memory systems. In *Proceedings of the Ninth ACM/IEEE International Parallel Processing Symposium (IPPS)*, April 1995.

[14] MIPS Technologies Inc. *MIPS R10000 Microprocessor User's Manual, Version 2.0*, December 1996.

[15] A. Moga and M. Dubois. The effectiveness of SRAM network caches in clustered DSMs. In *Proceedings of the Fourth Annual Symposium on High Performance Computer Architecture*, 1998.

[16] A. Nowatzyk, G. Aybay, M. Browne, E. Kelly, M. Parkin, B. Radke, and S. Vishin. The S3.mp scalable shared memory multiprocessor. In *Proceedings of the 1995 International Conference on Parallel Processing*, 1995.

[17] S. E. Perl and R.L. Sites. Studies of Windows NT performance using dynamic execution traces. In *Proceedings of the Second Symposium on Operating System Design and Implementation*, pages 169–184, October 1996.

[18] 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 325–336, April 1994.

[19] V. Santhanam, E.H. Fornish, and W.-C. Hsu. Data prefetching on the HP PA-8000. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, pages 264–273, June 1997.

[20] A. Saulsbury, F. Pong, and A. Nowatzyk. Missing the memory wall: The case for processor/memory integration. In *Proceedings of the 23rd Annual Symposium on Computer Architecture*, pages 90–101, May 1996.

[21] A. Saulsbury, T. Wilkinson, J. Carter, and A. Landin. An argument for Simple COMA. In *Proceedings of the First Annual Symposium on High Performance Computer Architecture*, pages 276–285, January 1995.

[22] Sun Microsystems. Ultra Enterprise 10000 System Overview. http://www.sun.com/servers/datacenter/products/starfire.

[23] W. Weber, S. Gold, P. Helland, T. Shimizu, T. Wicki, and W. Wilcke. The mercury interconnect architecture: A cost-effective infrastructure for high-performance servers. In *SIGARCH97*, page ???, June 1997.

[24] S.C. Woo, M. Ohara, E. Torrie, J.P. Singh, and A. Gupta. The SPLASH-2 programs: Characterization and methodological considerations. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pages 24–36, June 1995.