Design Alternatives for Shared Memory Multiprocessors

John Carter, Chen-Chi Kuo, Ravindra Kuramkote, and Mark Swanson*

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

E-mail: {retrac,chenchi,kuramkot,swanson}@cs.utah.edu

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 five 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 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 adaptive 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.*

*Mark Swanson is now at Intel Corporation. Current email addresses: mark.r.swanson@intel.com
This research was supported in part by the Space and Naval Warfare Systems Command (SPAWAR) and the Advanced Research Projects Agency (ARPA), under SPAWAR contract No.#N0039-95-C-0018 and ARPA Order No.#B990. The views and conclusions contained herein are those of the authors and should not be interpreted as necessariy representing the official policies or endorsements, either expressed or implied, of DARPA, the Air Force Research Laboratory, or the US Government.

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 compared to message passing is that it is easier to program, because programmers are not forced to track the location of every piece of data that might be needed. However, naive exploitation of shared memory 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. 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 processor and reducing the average memory latency. Because of the growing gap between processor cycle times and main memory latencies, modern processors incorporate a variety of latency-tolerating features such as fine-grained multithreading, lockup free caches, split transaction busses, and out-of-order execution [1, 12]. 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 a proprietary processor, 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) coherence misses, and (iii) conflict/capacity misses, hereafter referred to simply as conflict misses. The frequency of cold and coherence 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 time 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 [17], SGI Origin 2000 [7] and Mercury Interconnect Architecture [18] have put considerable effort into reducing the remote memory latency by developing specialized high speed interconnects. As an alternative, the designers of STiNG [10] included a large DRAM network cache in the DSM controller to reduce the number of remote accesses. Simple-COMA (S-COMA) [16] proponents have espoused using part of the local DRAM memory as remote memory page cache. Recently researchers have suggested extending S-COMA to adapt to an hybrid architecture that combines the best properties of both the CC-NUMA and S-COMA memory models [4, 6, 13].

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 the 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 [7, 8, 17], CC-NUMA extended to include either a DRAM RAC [10] (DRAC) or an SRAM RAC [9] (SRAC), pure Simple COMA [16] (S-COMA), and a hybrid CC-NUMA/S-COMA architecture [4, 6, 13] we call AS-COMA [6] or *adaptive S-COMA*. Using detailed execution-driven simulation, we examined these five architectures using two interconnects of significantly differing performance characteristics on five 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 RAC and *adaptive 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 lags 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*.

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. Finally, we draw conclusions and discuss possible future work in Section 5.

2 Design

In this section, we discuss the organization of the DSM machines that we are going evaluate: CC-NUMA, CC-NUMA extended with a 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, as described below. 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 consist 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 that is used to store directory state. RAC is present only in the two RAC configurations, while the page cache state region is present only in the S-COMA and AS-COMA models.

The overhead of remote overhead accesses can be represented as: $(N_{pagecache} * T_{pagecache}) + (N_{srac} * T_{srac}) + (N_{drac} * T_{drac}) + (N_{rem} * T_{rem}) + (N_{cold} * T_{rem}) + T_{overhead}$. $N_{pagecache}$, N_{srac}, N_{drac} and N_{rem} represent the number of conflict misses that were satisfied by the page cache, SRAC, DRAC and remote memory. N_{cold} represents the number of cold misses induced by flushing and remapping pages, and this is zero in the three CC-NUMA models. $T_{pagecache}$, T_{srac}, T_{drac} and T_{rem} represent the latency of fetching a line from the page cache, SRAC, DRAC and remote memory. $T_{overhead}$ represents the kernel overhead experienced by the S-COMA and AS-COMA models.

Table 1 summarizes the remote memory overhead and where one can invest to reduce it, 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.

2.2 CC-NUMA

In CC-NUMA [7], 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 [4].

The conflict miss cost in the CC-NUMA model is represented by $(N_{rem} * T_{rem})$, 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 (T_{rem}). Such an investment also reduces the cold and coherent access overhead, helping programs dominated by any of these three miss types.

2.3 CC-NUMA with RAC

In the RAC model [9, 10], a non-inclusive secondary cache for remote data is added to the DSM controller to help reduce conflict miss costs by reducing N_{rem}. The RAC model operates just as CC-NUMA except that a line that is fetched from a remote node is also stored in the RAC. If the line is evicted from 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.

In the SRAC and DRAC models, the overhead is given by $(N_{srac} * T_{srac}) + (N_{rem} * T_{rem})$ and $(N_{drac} * T_{drac}) + (N_{rem} * T_{rem})$, respectively. The remote overhead can be reduced either by increasing the RAC size, which reduces N_{rem}, or by reducing T_{rem}.

2.4 S-COMA

In the S-COMA model [15, 16], the DSM controller and operating system cooperate to provide access to remotely homed data. A mapping from a global virtual address to a *local* physical address is created at the first page fault to that shared memory page. 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 a 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 completely eliminate N_{rem} when the *memory pressure* on a node is low [6]. 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 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 con-

Model	Remote Overhead	Performance Factors	Storage Cost	Complexity
CC-NUMA	$(N_{rem} * T_{rem})$	Network speed	None	None
CC-NUMA (SRAC)	$(N_{srac} * T_{srac})+$ $(N_{rem} * T_{rem})$	1. Network speed 2. SRAM size and associativity	SRAM	Controller for SRAM
CC-NUMA (DRAC)	$(N_{drac} * T_{drac})+$ $(N_{rem} * T_{rem})$	1. Network speed 2. DRAM size and associativity	DRAM	Controller for DRAM
S-COMA	$(N_{pagecache} * T_{pagecache})+$ $(N_{cold} * T_{rem}) + T_{overhead}$	1. Network speed 2. Software overhead	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	$(N_{pagecache} * T_{pagecache})+$ $(N_{rem} * T_{rem})+$ $(N_{cold} * T_{rem}) + T_{overhead}$	1. Network speed 2. Software overhead	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

Table 1. Remote Memory Overhead and Complexity of Various Models

tention 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. Also, there is a slight increase in the complexity of the cache controller since it must examine the valid bit and also must 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 S-COMA model, the conflict miss cost is represented by $(N_{pagecache} * T_{pagecache}) + (N_{cold} * T_{rem}) + T_{overhead}$. Up to a certain application-dependent memory pressure threshold, page remapping does not occur and N_{cold} is zero. As the memory pressure increases beyond this threshold, N_{cold} 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 ($T_{overhead}$) 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 [6] is a hybrid architecture based on the S-COMA model [16]. Initially, it maps all pages in S-COMA mode. 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. AS-COMA uses a refetch counter mechanism similar to RNUMA [4] to identify hot pages. The directory controller maintains one counter per remote node for each of its home pages. Every time a remote node refetches a line from a page, the corresponding refetch counter is incremented. When the counter crosses a threshold, the directory controller interrupts the node refetching the data to inform it that the page is *hot*. Once the AS-COMA page cache fills up and pages must be evicted to make room for newly identified hot pages, a pageout daemon is run periodically and selects *cold* pages in the page cache for eviction. Cold pages are those that have not been actively referenced - they can be detected using coventional TLB reference bit techniques. 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 memory page corresponding to each evicted page is then mapped to the global physical page back at the home node. Subsequent cache line misses to such pages are satisfied as in CC-NUMA.

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. However, AS-COMA entails all of the implementation costs of S-COMA and more. First, there is another slight increase in the complexity of the directory 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 S-COMAs for memory pressures low enough to replicate all of the remote data accessed by a node. At these low memory pressures, the conflict miss cost will be $N_{pagecache} * T_{pagecache}$. When the memory pressure increases beyond this point, page remapping begins and the conflict miss cost becomes $(N_{pagecache} * T_{pagecache}) + (N_{rem} * T_{rem}) + (N_{cold} * T_{rem}) + T_{overhead}$, where N_{rem} and N_{cold} will remain small up to some higher memory pressure threshold where the page cache is no longer large enough to hold all hot pages. After this point, AS-COMA will begin to map hot pages into CC-NUMA mode. As memory pressure approaches 100%, the conflict miss cost of an ideal AS-COMA will degrade monotonically to that of CC-NUMA, $N_{rem} * T_{rem}$. Realizable AS-COMA models, however, will fare worse than CC-NUMA at high memory pressures due to the extra kernel overhead ($T_{overhead}$) incurred before the system stabilizes. However, due to its backoff algorithm, the kernel overhead of AS-COMA will remain far smaller than that of pure S-COMA.

AS-COMA differs from other hybrid approaches [4, 13] 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 [6].

3 Performance Evaluation

3.1 Experimental Setup

All experiments were performed using an execution-driven simulation of the Avalanche architecture [3, 14]. 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 modeled physical page size is 4 kilobytes. 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 [11] 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 Table 2. 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. 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 [5, 7]. 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, Table 2 also 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 higher than this minimum because of contention for various resources (bus, memory banks, networks, etc.) that we accurately model.

3.2 Benchmark Programs

We used five programs from the SPLASH-2 benchmark suite [19] in our study: `radix`, `fft`, `lu`, `barnes`, and `ocean`. Table 3 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 five applications. We selected a proces-

Component	Characteristics
L1 Cache	Size: 8-kilobytes. 32 byte lines, direct-mapped, virtually indexed, physically tagged, non-blocking, 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) Local memory: 58 cycles, remote memory (fast network): 147 cycles, and (slow network): 491 cycles

Table 2. Cache and Network Characteristics

Program	Input parameters	Home Pages (per node)	Maximum Remote Pages	Ideal Pressure	S-COMA Metadata	AS-COMA Metadata
`radix`	1M Keys, Radix = 1024	259	1306	17	.38 - 17	.55 - 24
`FFT`	256K Points, tuned for cache sizes	781	990	44	1.1 - 13	1.4 - 15
`LU`	1024x1024 matrix, 16x16 blocks, contiguous	514	405	56	.76 - 5	.93 - 6
`barnes`	16K particles	102	552	16	.15 - 7	.22 - 10
`ocean`	258x258 ocean	473	356	57	.69 - 4	1 - 6

Table 3. Programs Data Sizes and Metadata Requirements (kilobytes) in Experiments

sor 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 set and the amount of caching it has available reasonable compared to real systems.

The *Maximum Remote Pages* column presents the maximum number of remote pages that 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.

Table 3 also shows the amount of memory required to store S-COMA or AS-COMA 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 Table 1). 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 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

The left columns in Figures 1 and 2 present the relative execution time of our five 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, or performing *kernel* operations, such as page replacement or process synchronization. All results are for the parallel phase of the applications.

The bars in Figures 1 and 2 represent pure CC-NUMA, pure S-COMA, CC-NUMA augmented with an 4-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 (`F`: fast network and `S`: slow 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 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.

The right columns in Figures 1 and 2 illustrate 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 the left columns of Figures 1 and 2, we can divide the five 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` and `lu`), 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 the right column of Figure 1 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 pure 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`, and `lu`). `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 the right column of Figure 1 - 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 CC-NUMA variants for the same reason.

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 3). Even though `radix` accessed more remote pages than any other program, as illustrated in Table 3, 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 1 through 2:

- 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 with a fast network, the addition of a modest-sized DRAC can improve performance of a CC-NUMA machine 5-50% (e.g., `barnes`, `lu`, and `radix`). This result implies that the designers of the next generation CC-NUMA architectures should seriously consider adding a DRAC to their system even while using a fast 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 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 [10], and employing a fast special-purpose network, as is done in the SGI Origin 2000 [7], 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 [4, 13] 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. 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, Sept. 1990.

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

[3] J. Carter, A. Davis, R. Kuramkot, C.-C. Kuo, M. Swanson, and L. Stoller. Avalanche: A communication and memory architecture for scalable parallel computing. Technical

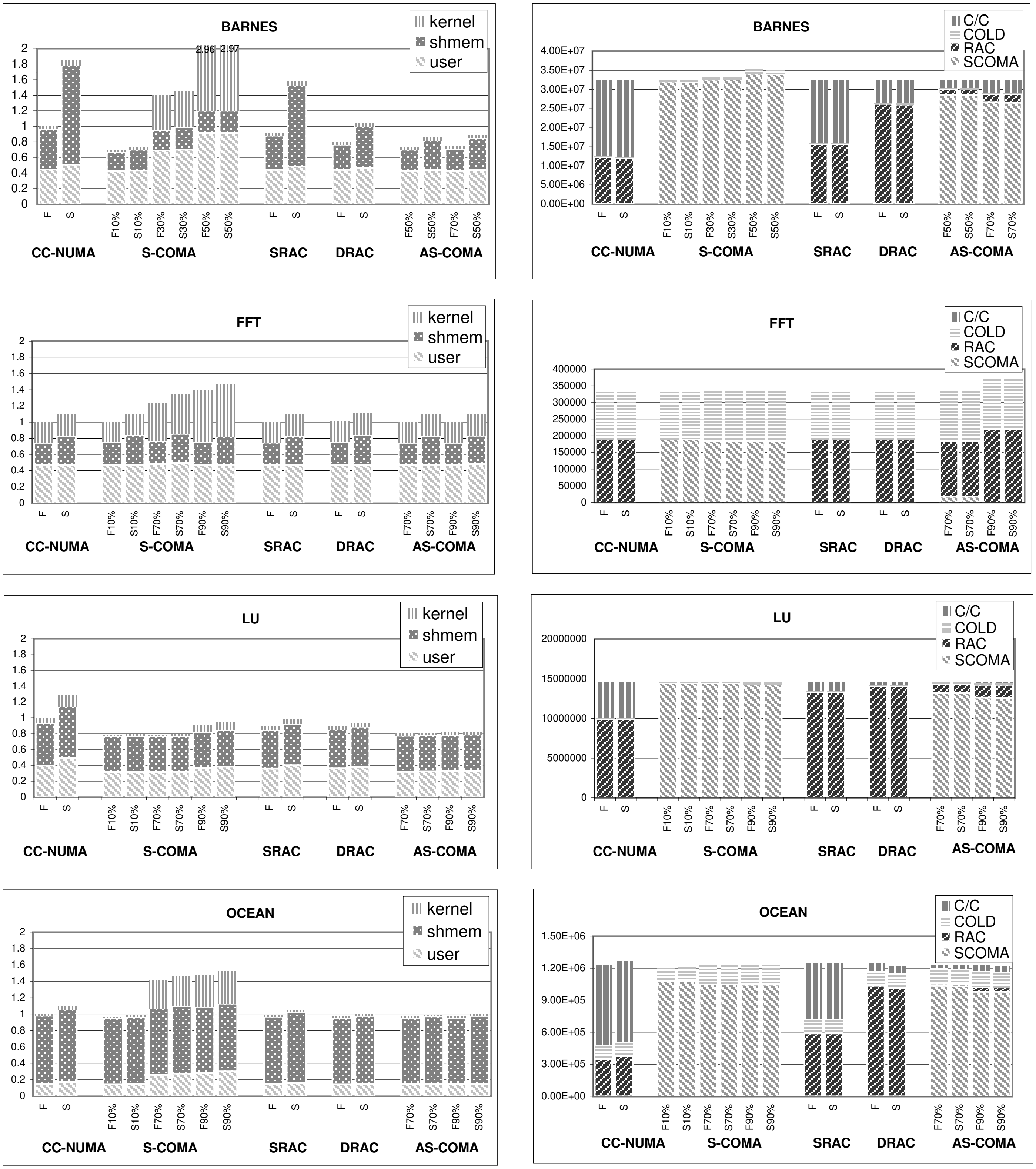
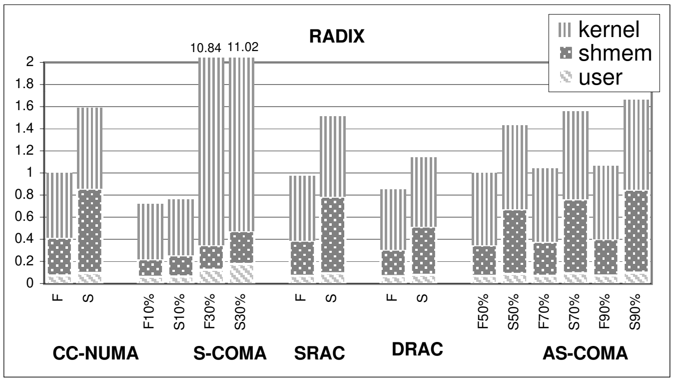


Figure 1. Relative Execution Times (Left Column) and Where Cache Misses Were Satisfied (Right Column) for `barnes`, `fft`, `lu` and `ocean`



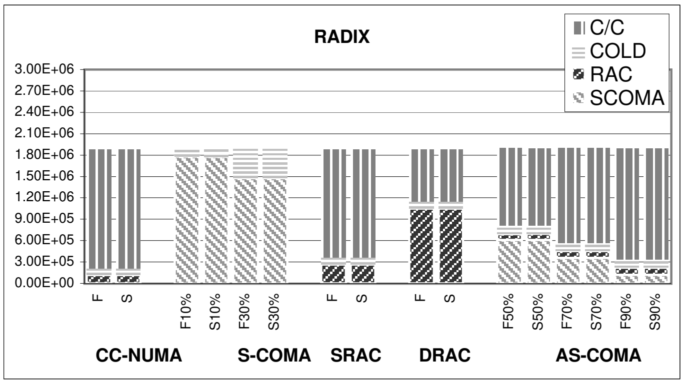


Figure 2. Relative Execution Times (Left Column) and Where Cache Misses Were Satisfied (Right Column) for `radix`

Report UUCS-95-022, University of Utah - Department of Computer Science, 1995.

[4] B. Falsafi and D. 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.

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

[6] C.-C. Kuo, J. Carter, R. Kuramkote, and M. Swanson. AS-COMA: An adaptive hybrid shared memory architecture. In *1998 International Conference on Parallel Processing*, Aug. 1998.

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

[8] 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.

[9] 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, Mar. 1992.

[10] 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.

[11] M. Marchetti, L. Kontothonassis, R. Bianchini, and M. 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)*, Apr. 1995.

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

[13] 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.

[14] R. Nalumasu and G. Gopalakrishnan. Deriving efficient cache coherence protocols through refinement. In *Formal Methods in Parallel Programming: Theory and Applications*, Orlando, Florida, USA, Apr. 1998.

[15] S. Reinhardt, J. Larus, and D. Wood. Tempest and Typhoon: User-level shared memory. In *Proceedings of the 21st Annual International Symposium on Computer Architecture*, pages 325–336, Apr. 1994.

[16] 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, Jan. 1995.

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

[18] 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*, June 1997.

[19] S. Woo, M. Ohara, E. Torrie, J. 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.