ASCOMA : An Adaptive Hybrid Shared Memory Architecture

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

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

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

Abstract

Scalable shared memory multiprocessors traditionally use either a cache coherent non-uniform memory access (CC-NUMA) or simple cache-only memory architecture (S-COMA) memory architecture. Recently, hybrid architectures that combine aspects of both CC-NUMA and S-COMA have emerged. In this paper, we present two improvements over other hybrid architectures. The first improvement is a page allocation algorithm that prefers S-COMA pages at low memory pressures. Once the local free page pool is drained, additional pages are mapped in CC-NUMA mode until they suffer sufficient remote misses to warrant upgrading to S-COMA mode. The second improvement is a page replacement algorithm that dynamically backs off the rate of page remappings from CC-NUMA to S-COMA mode at high memory pressure. This design dramatically reduces the amount of kernel overhead and the number of induced cold misses caused by needless thrashing of the page cache. The resulting hybrid architecture is called adaptive S-COMA (AS-COMA). AS-COMA exploits the best of S-COMA and CC-NUMA, performing like an S-COMA machine at low memory pressure and like a CC-NUMA machine at high memory pressure. AS-COMA outperforms CC-NUMA under almost all conditions, and outperforms other hybrid architectures by up to 17% at low memory pressure and up to 90% at high memory pressure.

*Mark Swanson is now at Intel Corporation. Current email addresses: Mark_R_Swanson@ccm.dp.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 as 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 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 problem that shared memory is designed to eliminate. Therefore, it is important to develop memory architectures that reduce the overhead of remote memory access.

Remote memory overhead is governed by three issues: (i) the number of cycles required to satisfy each remote memory request, (ii) the frequency with which remote memory accesses occur, and (iii) the software overhead of managing the memory hierarchy. The designers of high-end commercial DSM systems such as the SUN UE10000 [12] and SGI Origin 2000 [5] have put considerable effort into reducing the remote memory latency by developing specialized high speed interconnects. These efforts can reduce the ratio of remote to local memory latency to as low as 2:1, but they require expensive hardware available only on high-end servers costing hundreds of thousands of dollars. In this paper, we concentrate on the second and third issues, namely reducing the frequency of remote memory accesses while ensuring that the software overhead required to do this remains modest. Previous studies have tended to ignore the impact of software overhead [3, 9, 10], but our findings indicate that the effect of this factor can be dramatic.

Scalable shared memory multiprocessors traditionally use either a *cache coherent non-uniform memory access* (CC-NUMA) architecture [5, 6, 12] or a *simple cache-only memory architecture* (S-COMA) [10]. In a CC-NUMA, the amount of remote shared data that can be replicated on a node is limited by the size of a node's processor cache(s) and *remote access cache* (RAC) [6]. S-COMA architectures employ any unused DRAM on a node as a cache for remote data [10]. However, the performance of pure S-COMA machines is heavily dependent on the *memory pressure* of a particular application. Put simply, memory pressure is a measure of the amount of physical memory in a machine required to hold an application's instructions and data. A 20% memory pressure indicates that 20% of a machine's pages must be used to hold the initial (home) copy of the application's instructions and data. Although this ability to cache remote data in local memory can dramatically reduce the number of remote memory operations, page management can be expensive.

Recently, hybrid architectures that combine aspects of both CC-NUMA and S-COMA have emerged, such as the Wisconsin *reactive CC-NUMA* (R-NUMA) [3] and the USC *victim cache NUMA* (VC-NUMA) [9]. Intuitively, these hybrid systems attempt to map the remote pages for which there are the highest number of conflict misses to local S-COMA pages, so-called *hot* pages, thereby eliminating the greatest number of expensive remote operations. All other remote pages are mapped in CC-NUMA mode. Ideally, such systems would exploit unused available DRAM for caching without penalty but the proposed implementations fail to achieve this goal under certain conditions.

In this paper, we present two improvements over R-NUMA and VC-NUMA. The first improvement is a page allocation algorithm the prefers S-COMA pages at low memory pressures. Once the local free page pool is drained, additional pages are initially mapped in CC-NUMA mode until they suffer sufficient remote misses to warrant upgrading to S-COMA mode. The second improvement is a page replacement algorithm that dynamically backs off the rate of page remappings between CC-NUMA and S-COMA mode at high memory pressure. This design dramatically reduces the amount of kernel overhead and the number of induced cold misses caused by needless thrashing of the page cache. The resulting hybrid architecture is called *adaptive S-COMA* (AS-COMA).

We used detailed execution-driven simulation to evaluate a number of AS-COMA design tradeoffs and then compared the resulting AS-COMA design against CC-NUMA, pure S-COMA, R-NUMA, and VC-NUMA. We found that AS-COMA's hybrid design provides the best behavior of both CC-NUMA and S-COMA. At low memory pressures, AS-COMA acts like S-COMA and outperforms other hybrid architectures by up to 17%. At high memory pressures, AS-COMA avoids the performance dropoff induced by thrashing and aggressively converges to CC-NUMA performance, thereby outperforming the other hybrid architectures by up to 90%. In addition, AS-COMA outperforms CC-NUMA under almost all conditions, and at its worst only underperforms CC-NUMA by 5%.

The remainder of this paper is organized as follows. In Section 2 we describe the basics of all scalable shared memory architectures, followed by an in-depth description of existing DSM models. Section 3 presents the design of our proposed AS-COMA architecture. We describe our simulation environment, test applications, and experiments in Section 4, and present the results of these experiments in Section 5. Finally, we draw conclusions in Section 6.

2. Background

In this section, we discuss the organization of the existing DSM architectures: CC-NUMA, S-COMA, R-NUMA, and VC-NUMA.

2.1. Directory-based DSM Architectures

All of the shared memory architectures that we consider share a common basic design, as described below. Individual nodes are composed of one or more commodity microprocessors with private 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 a typical 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 of the design alternatives that we explore, the local storage contains DRAM that is used to store directory state.

The remote access overhead of these architectures can be represented as: $(N_{pagecache} * T_{pagecache}) + (N_{remote} * T_{remote}) + (N_{cold} * T_{remote}) + T_{overhead}$. $N_{pagecache}$ and N_{remote} represent the number of conflict misses that were satisfied by the page cache or remote memory, respectively. N_{cold} represents the number of cold misses induced by flushing and remapping pages, and thus is zero only in the CC-NUMA model. $T_{pagecache}$ and T_{remote} represent the latency of fetching a line from the local page cache or remote memory, respectively. $T_{overhead}$ represents the software overheads of the S-COMA and the hybrid models to support page remapping.

2.2. CC-NUMA

In CC-NUMA [5, 6, 12], the first page access on each node to a particular page causes a page fault, at which time the local TLB and page table are loaded with a page translation to the appropriate *global* physical page. The home node of each page can be determined from its physical address. When the local processor suffers a cache miss to a line in a remote page, the DSM controller forwards the memory request to the memory's home node, incurring a significant access delay. Remote data can only be cached in the processor cache(s) or an optional remote access cache (RAC) on the DSM controller. Applications that suffer a large number of conflict misses to remote data, e.g., due to the limited amount of caching of remote data, perform poorly on CC-NUMAs [3]. Unfortunately, these applications are fairly common [3, 10]. Careful page allocation [1, 7], migration [13], or replication [13] can alleviate this problem by selecting or modifying the choice of home node for a given page of data, but these techniques have to date only been successful for read-only or non-shared pages.

The conflict miss cost in the CC-NUMA model is represented by $(N_{remote} * T_{remote})$, that is, all misses to shared memory with a remote home must be remote misses. To reduce this overhead, designers of some such systems have adopted high speed interconnects to reduce T_{remote} [5, 12].

2.3. S-COMA

In the S-COMA model [10], the DSM controller and operating system cooperate to provide access to remotely homed data. S-COMA's aggressive use of local memory to replicate remote shared data can completely eliminate N_{remote} when the memory pressure on a node is low. However, pure S-COMA's performance degrades rapidly for some applications as memory pressure increases. Because *all* remote data *must* be mapped to a local physical page before it can be accessed, there can be heavy contention if the number of local physical pages available for S-COMA page replication is small. Under these circumstances, thrashing occurs, not unlike thrashing in a conventional VM system. Given the high cost of page replacement, this can lead to dismal performance.

In the S-COMA model, the conflict miss cost is represented by $(N_{pagecache} * T_{pagecache}) + (N_{cold} * T_{remote}) + T_{overhead}$. When memory pressure is low enough that all of the remote data a node needs can be cached locally, page remapping does not occur and both N_{cold} and $T_{overhead}$ are zero. As the memory pressure increases, and thus more remote pages are accessed by a node than can be cached locally, N_{cold} and $T_{overhead}$ increase due to remapping. N_{cold} increases because the contents of any pages that are replaced from the local page cache must be flushed from the processor cache(s). Subsequent accesses to these pages will suffer cold misses in addition to the cost of remapping. An even worse problem is that as memory pressure approaches 100%, the time spent in the kernel flushing and remapping pages ($T_{overhead}$) skyrockets. Sources of this overhead include the time spent context switching between the user application and the pageout daemon, flushing blocks from the victim page(s), and remapping pages.

2.4. Hybrid DSM Architectures

Two hybrid CC-NUMA/S-COMA architectures have been proposed: R-NUMA [3] and VC-NUMA [9]. We describe these architectures in this section.

The basic architecture of an R-NUMA machine [3] is that of a CC-NUMA machine. However, unlike CC-NUMA, which "wastes" local physical memory not required to hold home pages, R-NUMA uses this otherwise unused storage to cache frequently accessed remote pages, as in S-COMA. This mechanism requires a number of modest modifications to a conventional CC-NUMA's DSM engine and operating system, as described below.

In addition to its normal CC-NUMA operation, the directory controller in an R-NUMA machine maintains an array of counters that tracks for each page the number of times that each processor has refetched a line from that page, as follows. Whenever a directory controller receives a request for a cache line from a node, it checks to see if that node is already in the copyset of nodes for that line. If it is, this request is a *refetch* caused by a conflict miss, and not a coherence or cold miss, and the node's refetch counter for this page is incremented. The per-page/per-node counter is used to determine which CC-NUMA pages are generating frequent remote refetches, and thus are good candidates to be mapped to an S-COMA page on the accessing node. When a refetch counter crosses a configurable threshold (e.g., 64), the directory controller piggybacks an indication of this event with the data response. This causes the DSM engine on the requesting node to interrupt the processor with an indication that a particular page should be remapped to a local S-COMA page. In a recent study [3], R-NUMA's flexibility and intelligent selection of pages to map in S-COMA mode caused it to outperform the best of pure CC-NUMA and pure S-COMA by up to 37% on some applications.

Although R-NUMA frequently outperforms both CC-NUMA and S-COMA, it was also observed to perform as much as 57% worse on some applications [3]. This poor performance can be attributed to two problems. First, R-

NUMA initially maps all pages in CC-NUMA mode, and only upgrades them to S-COMA mode after some number of remote refetches occur, which introduces needless remote refetches when memory pressure is low. Second, R-NUMA always upgrades pages to S-COMA mode when their refetch threshold is exceeded, even if it must evict another hot page to do so. When memory pressure is high, and the number of hot pages exceeds the number of free pages available for caching them, this behavior results in frequent expensive page remappings for little value. This leads to performance worse than CC-NUMA, which never remaps pages.

VC-NUMA [9] treats its RAC as a victim cache for the processor cache(s). However, this solution requires significant modifications to the processor cache controller and bus protocol. The designers of VC-NUMA also noticed the tendency of hybrid models to thrash at high memory pressure and suggested a thrashing detection scheme to address the problem. Their scheme requires a local refetch counter per S-COMA page, a programmable *break even* number that depends on the network latency and overhead of relocating pages, and an *evaluation threshold* that depends on the total number of free S-COMA pages in the page cache. Although VC-NUMA frequently outperforms R-NUMA, the study did not isolate the benefit of the thrashing detection scheme from that of the integrated victim cache.

In these hybrid models, the conflict miss cost is represented by $(N_{pagecache} * T_{pagecache}) + (N_{remote} * T_{remote}) + (N_{cold} * T_{remote}) + T_{overhead}$. $N_{pagecache}$ and N_{remote} closely depend on the relocation mechanisms. Remappings between CC-NUMA and S-COMA modes account for the increased cold miss rate (N_{cold}), as described earlier. $T_{overhead}$ is the software overhead required for the kernel to handle interrupts, flush pages, and remap pages.

When there are plentiful free local pages, the difference between the hybrid models and S-COMA is that S-COMA does not suffer from as many initial conflict misses, nor does it pay for page remapping. In such a case, the relative costs between the two models can be represented as:

$$N_{remote.hybrid} + N_{cold.hybrid} \gg N_{cold.scoma} \approx 0, \quad (1)$$

$$T_{overhead.hybrid} > T_{overhead.scoma} \approx 0, \quad (2)$$

$$N_{pagecache.scoma} > N_{pagecache.hybrid} \quad (3)$$

As the memory pressure increases, R-NUMA and VC-NUMA suffer from the same problems as pure S-COMA, although to a lesser degree. Even hot pages already in the page cache begin to be remapped. In a worst case, the relative cost between the hybrid models and CC-NUMA under high memory pressure can be represented as:

$$N_{remote.hybrid} + N_{cold.hybrid} > N_{remote.ccnuma}, \quad (4)$$

$$T_{overhead.hybrid} \gg T_{overhead.ccnuma} \approx 0. \quad (5)$$

Relations (1), (2) and (3) suggest that one way to improve the hybrid models at low memory pressure is to accelerate their convergence to S-COMA. Likewise, relations (4) and (5) suggest that performance can be improved by throttling CC-NUMA $\leftrightarrow$ S-COMA transitions at high memory pressure. Unlike S-COMA, in which remapping is required for the architecture to operate correctly, the hybrid architectures can choose to stop remapping and leave pages in CC-NUMA mode.

In summary, the performance of hybrid S-COMA/CC-NUMA architectures is significantly influenced by the memory pressure induced by a particular application. Since it is common for users to run the largest applications they can on their hardware, the performance of an architecture at high memory pressures is particularly important. Therefore, it is crucial to conduct performance studies of S-COMA or hybrid architectures across a broad spectrum of memory pressures. An improved hybrid architecture, motivated by the analysis above, that performs well regardless of memory pressure is discussed in the following section.

3. Adaptive S-COMA

At low memory pressure, S-COMA outperforms CC-NUMA, but the converse is true at high memory pressure [10]. Thus, our goal when designing AS-COMA was to develop a memory architecture that performed like pure S-COMA when memory for page caching was plentiful, and like CC-NUMA when it is not.

To exploit S-COMA's superior performance at low memory pressures, AS-COMA initially maps pages in S-COMA mode. Thus, when memory pressure is low, AS-COMA will suffer no remote conflict or capacity misses, nor will it pay the high cost of remapping (i.e., cache flushing, page table remapping, TLB refill, and induced cold misses). Only when the page cache becomes empty does AS-COMA begin remapping.

Like the previous hybrid architectures, AS-COMA reacts to increasing memory pressure by evicting "cold" pages from, and remapping "hot" pages to, the local page cache. However, what sets AS-COMA apart from the other hybrid architectures is its ability to adapt to differing memory pressures to fully utilize the large page cache at low memory pressures and to avoid thrashing at high memory pressures. It does so by dynamically adjusting the *refetch threshold* that triggers remapping, increasing it when it notices that memory pressure is high.

AS-COMA uses the kernel's VM system to detect thrashing, as follows. The kernel maintains a pool of free local pages that it can use to satisfy allocation or relocation requests. The pageout daemon attempts to keep the size of this pool between *free_target* and *free_min* pages. Whenever the size of the free page pool falls below *free_min* pages, the pageout daemon attempts to evict enough "cold" pages to refill the free page pool to *free_target* pages. Only S-COMA pages are considered for replacement. To replace a page, its valid blocks are flushed from the processor cache, and then its corresponding global virtual address is remapped to its home physical address. *Cold* pages are detected using a *second chance* algorithm: the TLB reference bit associated with each S-COMA page is reset each time it is considered for eviction by the pageout daemon. If the reference bit is zero when the pageout daemon next runs, the page is considered cold.

Under low to moderate memory pressure, allocation or relocation requests can be performed immediately because there will be pages in the free page pool. However, at heavy memory pressure, the pageout daemon will be unable to find sufficient cold pages to refill the free page pool. Whenever the pageout daemon is unable to reclaim at least *free_target* free pages, AS-COMA begins allocating pages in CC-NUMA mode under the assumption that local memory can not accommodate the application's entire working set. In addition, it raises the refetch threshold by a fixed amount to reduce the rate at which "equally-hot" pages in the page cache replace each other. It also increases the time between successive invocations of the pageout daemon. Should the number of hot pages drop, e.g., because of a phase change in the program that causes a number of hot pages to grow cold, the pageout daemon will detect it by seeing an increase in the number of cold pages. At this point, it can reduce the refetch threshold.

Using this backoff scheme, the rate at which destructive flushing and remapping occurs is decreased, as is the number of cold misses induced by remapping. In addition, the frequency at which the pageout daemon is invoked is reduced, which eliminates context switches and pageout daemon execution time. Overall, we found this back pressure on the replacement mechanism to be extremely important. As will be shown in Section 5, it alleviates the performance slowdowns experienced by R-NUMA or VC-NUMA when memory pressure is high.

AS-COMA's conflict miss cost is identical to SCOMA's when there are enough local free pages to accommodate the application's working set. In such cases, the remote refetch cost of AS-COMA will be close to $(N_{pagecache} * T_{pagecache})$. Until memory pressure gets high, N_{remote} will grow slowly. Eventually the page cache will no longer be large enough to hold all hot pages. Ideally AS-COMA's performance would simply degrade smoothly to that of CC-NUMA, $(N_{remote} * T_{remote})$, as memory pressure approaches 100%. Realizable AS-COMA models will fare somewhat worse due to the extra kernel overhead incurred before the system stabilizes. Nevertheless, AS-COMA is able to converge rapidly to either S-COMA or CC-NUMA mode, depending on the memory pressure.

4. Performance Evaluation

4.1. Experimental Setup

All experiments were performed using an execution-driven simulation of the HP PA-RISC architecture called Paint (PA-interpreter)[11]. Our simulation environment includes detailed simulation modules for a first level cache, system bus, memory controller, and DSM engine. Note that our network model only accounts for input and output ports contention. The simulation environment also provides a multiprogrammed process model with support for operating system code, so the effects of OS/user code interactions are modeled. It 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. All three hybrid architectures we study adopt BSD4.4's page allocation mechanism and paging policy [8] with minor modifications. *Free_min* and *free_target* (see Section 3) were set to 5% and 7% of total memory, respectively. We extended the first touch allocation algorithm [7] to distribute home pages equally.

The modeled processor and DSM engine are clocked at 120MHz. The system bus modeled is HP's Runway bus, which is also 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 1.

For most of the SPLASH2 applications we studied, the data sets provided have a primary working set that fits in an 8-kbyte cache[14]. We, therefore, model a single 8-kilobyte direct-mapped processor cache to compensate for the small size of the data sets, which is consistent with previous studies of hybrid architectures[3, 9].

We modeled a 4-bank main memory controller that can supply data from local memory in 58 cycles. The size of main memory and the amount of free memory used for page caching was varied to test the different models under varying memory pressures.

We used a sequentially-consistent write-invalidate consistency protocol with a coherence unit of 128 bytes (4 lines). Our CC-NUMA and hybrid models are not "pure," as we employ a 128-byte RAC containing the last remote data received as part of performing a 4-line fetch. An initial relocation threshold of 32, the number of remote refetches required to initiate a page remapping, is used in all three hybrid architectures. The relocation thresholds were incremented by 8 whenever thrashing is detected by AS-COMA's software scheme or by VC-NUMA's hardware scheme. VC-NUMA uses a breakeven number of 16 for its thrashing detection mechanism. We did not simulate VC-NUMA's victim-cache behavior, because we considered the use of non-commodity processors or busses to be beyond the scope

Component	Characteristics
L1 Cache	Size: 8-kilobytes. 32 byte lines, direct-mapped, virtually indexed, physically tagged, non-blocking, up to one outstanding miss, write back, 1-cycle hit latency
RAC	128 byte lines, direct-mapped, non-inclusive, non-blocking, up to one outstanding miss, 23-cycle hit latency.
Networks	1 cycle propagation, 2X2 switch topology, port contention (only) modeled Fall through delay: 4 cycles (local and remote memory access latencies - 58 and 147 cycles)

Table 1. Cache and Network Characteristics

of this study. Thus, the results reported for VC-NUMA are only relevant for evaluating its relocation strategy, and not the value of treating the RAC as a victim cache[9].

4.2. Benchmark Programs

We used six programs to conduct our study: `barnes`, `fft`, `lu`, `ocean`, and `radix` from the SPLASH-2 benchmark suite [14] and `em3d` from a shared memory implementation of the Split-C benchmark [2]. Table 2 shows the inputs used for each test program. The column labeled *Total Home pages* indicates the total number of shared data pages initially allocated at all nodes. These numbers indicate that each node manages from 0.5 megabytes (`barnes`) to 2 megabytes (`lu, em3d, and ocean`) of home data.

The *Total Remote Pages* column indicates the maximum number of remote pages that are accessed by all nodes for each application, which gives an indication of the size of the application's global working set. The *Ideal pressure* column is the memory pressure below which 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 any kernel or page daemon overhead to remap pages.

Due to its small default problem size and long execution time, `lu` was run on just 4 nodes - all other applications were run on 8 nodes.

5. Results

Figure 1 shows the performance of CC-NUMA, S-COMA, and three hybrid CC-NUMA/S-COMA architectures (AS-COMA, VC-NUMA, R-NUMA) on the six applications. Each graph displays the execution time of the various architectures relative to CC-NUMA, and indicates where this time was spent by each program[1]. More performance statistics, e.g., where cache misses to shared data were satisfied, can be found in the technical report version of this paper [4]. We simulated the applications across a range of memory pressures between 10% and 90%. Only one result is shown for CC-NUMA, since it is not affected by memory pressure. As can be seen in the graphs, the relative performance of the different architectures can vary dramatically as memory pressures change. All results include only the parallel phase of the various programs.

5.1. Initial Allocation Schemes

We will first focus on the effect of the initial allocation policies. As shown in Table 2, the "ideal" memory pressure for the six applications ranged from 16% to 57%. Below this memory pressure, the local page cache is large enough to store the entire working set of a node. To isolate the impact of initially allocating pages in S-COMA, we simulated S-COMA and the hybrid architectures at a memory pressure of 10%, when no page remappings beyond any initial ones will occur. Table 2 shows the percentage of remote pages that are refetched at least 32 times (*Relocated Pages*), and thus will be remapped from CC-NUMA to S-COMA mode in R-NUMA or VC-NUMA, the total number of remote pages accessed (*Total Remote Pages*), and the percentage of remote pages eligible for relocation (*% of Relocated Pages*). This percentage ranges from under 1% in `fft` to over 95% in `lu` and `radix`.

First, to illustrate the importance of employing a hybrid memory architecture over a vanilla CC-NUMA architecture, examine Figure 1 at 10% memory pressures. At this low memory pressure, AS-COMA, like S-COMA, outperforms CC-NUMA by 20-35% for four of the applications (`lu`, `radix`, `barnes`, and `em3d`). Looking at the hybrid architectures in isolation, we can see that for `radix`, AS-COMA outperforms R-NUMA and VC-NUMA by 17%. In `radix`, the percentage and total number of remote pages that need to be remapped are both quite high, 98% and 10236 respectively. In the other applications, the initial page allocation policy had little impact on performance. There is no strong correlation between the number of pages

[1] *U-SH-MEM*: stalled on shared memory. *K-BASE*: performing essential kernel operations (i.e., those required by all architectures). *K-OVERHD*: performing architecture-specific kernel operations, such as remapping pages and handling relocation interrupts. *U-INSTR* and *U-LC-MEM*: performing user-level instructions or non-shared (local) memory operations. *SYNC*: performing synchronization operations.

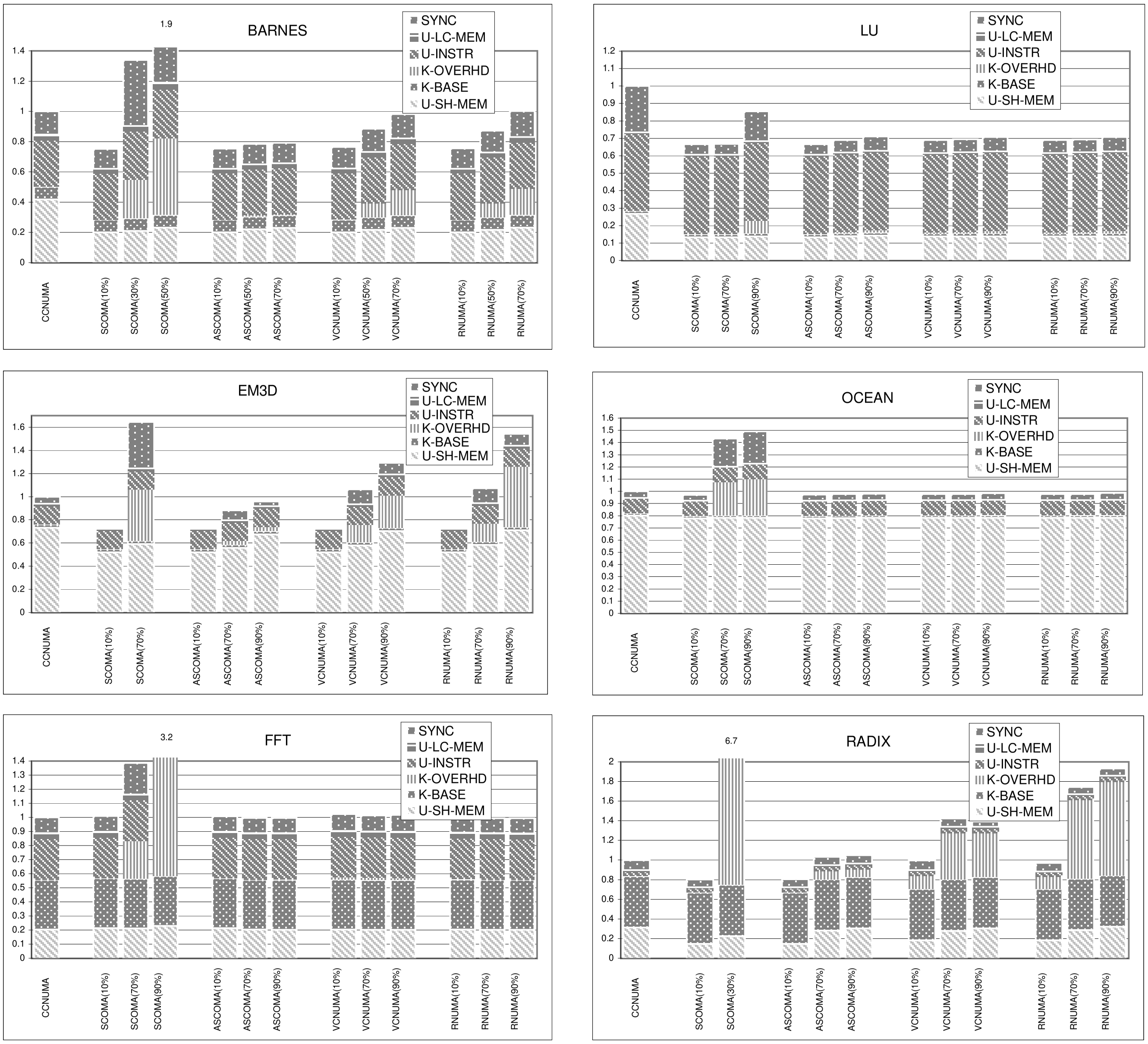

Figure 1. Relative Execution Time for `barnes`, `em3d`, `fft`, `lu`, `ocean`, **and** `radix`**.**

Program	Input parameters	Total Home Page	Total Remote Pages	Ideal Pressure	Relocated Pages	% of Relocated Pages
`barnes`	16K particles	816	4416	16%	3498	80%
`em3d`	40K nodes,15%remote,20 iters	3928	6224	39%	1868	29%
`FFT`	256K points	3120	10032	24%	5	0.05%
`LU`	1024x1024 matrix,16x16 blocks	2056	1620	56%	1606	99%
`ocean`	258x258 ocean	3784	2848	57%	569	20%
`radix`	1M Keys, Radix = 1024	2072	10448	17%	10236	98%

Table 2. Programs and Problem Sizes Used in Experiments and Their Shared Memory Pages Statistics

that need to be remapped and performance. We can observe a 5% performance benefit in `lu`, where the percentage of relocated remote pages is very high (99%), but the total number is fairly small (1606).

There are two primary reasons why the initial allocation policy did not have a stronger impact on performance. First, our interrupt and relocation operations are highly optimized, requiring only 2000 and 6000 cycles, respectively, to perform. Thus, the impact of the unnecessary remappings and flushes is overwhelmed by other factors. Second, as an artifact of our experimental setup, the initial remappings for several applications were not included in the performance results, as they took place before the parallel phase when our measurements are taken. This was the case for `barnes` and `em3d`. The final two applications, `fft` and `ocean`, only access a small number of remote pages enough times to warrant remapping, and thus the impact of initially mapping pages in S-COMA mode is negligible.

In summary, if memory pressure is low and local pages for replication are abundant, an S-COMA-preferred initial allocation policy can improve the performance hybrid architectures moderately by accelerating their convergence to pure S-COMA behavior. However, the performance boost is modest.

5.2. Thrashing Detection and Backoff Schemes

The performance of hybrid DSM architectures depends heavily on the memory pressure. Performance seriously degrades when the page cache cannot hold all "hot" pages and those pages start to evict one another. Intuitively, when this begins to occur, the memory system should simply treat the page cache as a place to store a *reasonable* set of hot pages, and stop trying to fine tune its contents since this tuning adds significant overhead. Previous studies have not considered the kernel overhead ($T_{overhead}$), but we found it to be very significant at high memory pressures. Once the page cache holds only hot pages, further attempts to refine its contents lead to thrashing, which involves unnecessary flushing of hot data, cache flushes, and induced cold misses. Since one hot page is replacing another, the benefit of this remapping is likely to be minimal compared to the cost of the remapping itself. As a result, the performance of a hybrid architecture will quickly drop below that of CC-NUMA if a mechanism is not put in place to avoid thrashing. As described in Section 3, the pageout daemon in AS-COMA detects thrashing when it cannot find cold pages to replace, at which point it reduces the rate of page remappings, going so far as to stop it completely if necessary. As can be seen in Figure 1, this can lead to significant performance improvements compared to R-NUMA and VC-NUMA under heavy memory pressure.

We can divide the six applications into two groups: (i) applications where there are sufficient remote conflict misses that handling thrashing effectively can lead to large performance gains (`barnes`, `em3d`, and `radix`), and (ii) applications in which minimal efforts to avoid thrashing are sufficient for handling high memory pressure (`fft`, `ocean`, and `lu`).

The behavior of `em3d` shows the danger of focusing solely on reducing remote conflict misses [4] when designing a memory architecture. As shown in Figure 1, the performance of `em3d` on the hybrid architectures is quite sensitive to memory pressure. R-NUMA outperforms CC-NUMA until memory pressure approaches 70%, after which time its performance drops quickly. CC-NUMA outperforms R-NUMA by 5% at 70% memory pressure and by 50% at 90%. Looking at the detailed breakdown of where time is spent, we can see that increasing kernel overhead is the culprit. In `em3d`, approximately 29% of remote pages, i.e., 230 pages, are eligible for relocation (see Table 2), but at 70% memory pressure there are only 210 free local pages. It turns out that for `em3d`, most of the remote pages ever accessed are in the node's working set, i.e., they are "hot" pages. Thus, above 70% memory pressure, R-NUMA begins to thrash and its performance degrades badly. This performance dropoff occurs even though there are significantly fewer remote conflict misses in R-NUMA than in CC-NUMA or AS-COMA [4]. The cost of constantly remapping pages between CC-NUMA and S-COMA mode and the increase in remote cold misses overwhelms the benefit of the reduced number of remote conflict misses. This behavior emphasizes the importance of detecting thrashing and reducing the rate of remappings when it occurs.

Recognizing this problem, VC-NUMA uses extra hardware to detect thrashing. However, its mechanism is not as effective as AS-COMA's. VC-NUMA starts to underperform CC-NUMA at the same memory pressure that R-NUMA does, 70%. While VC-NUMA outperforms R-NUMA by 22% at 90% memory pressure, it underperforms CC-NUMA by 27% and AS-COMA by 31%. In contrast, AS-COMA outperforms CC-NUMA even at 90% memory pressure, when the other hybrid architectures are thrashing. It does so by dynamically turning off relocation as it determines that this relocation has no benefits because it is simply replacing hot pages with other hot pages. This results in more remote conflict/capacity misses than the other hybrid architectures, but it reduces the number of cold misses caused by flushing pages during remapping and the kernel overhead associated with handling interrupts and remapping [4]. As a result, AS-COMA outperforms VC-NUMA by 31% and R-NUMA by 53% at 90% memory pressure. Moreover, despite having only a small page cache available to it and a remote working set larger than this cache, AS-COMA outperforms CC-NUMA.

Barnes exhibits very high spatial locality. It accesses large dense regions of remote memory, and thus can make good use of a local S-COMA page cache[2]. As shown in in Table 2, barnes's ideal memory pressure is 16%. Like em3d, most of the remote pages that are accessed are part of the working set and "hot" for long periods of execution. We observed that thrashing begins to occur at 50% memory pressure. As in em3d, R-NUMA reduces the number of remote conflict/capacity misses at high memory pressures, at the cost of increasing kernel overhead and remote cold misses. As a result, it is able to outperform CC-NUMA at low memory pressure, but is only able to break even by the time memory pressure reaches 70%. Similarly, VC-NUMA's backoff mechanism is not sufficiently aggressive at moderate memory pressures to stop the increase in kernel overhead or cold misses. In particular, VC-NUMA only checks its backoff indicator when an average of two replacements per cached page have occurred, which is not sufficiently often to avoid thrashing. As shown in the previous study [9], VC-NUMA does not significantly outperform R-NUMA until memory pressure exceeds 87.5%. Once again, AS-COMA's adaptive replacement algorithm detects thrashing as soon as it starts to occur, and the resulting backoff mechanism causes performance to degrade only slightly as memory pressure increases. As a result, it consistently outperforms CC-NUMA by 20% across all ranges of memory pressures, and outperforms the other hybrid architectures by a similar margin at high memory pressures.

Unlike barnes, radix exhibits almost no spatial locality. Every node accesses every page of shared data at some time during execution. As such, it is an extreme example of an application where fine tuning of the S-COMA page cache will backfire - each page is roughly as "hot" as any other, so the page cache should simply be loaded with some reasonable set of "hot" pages and left alone. With an ideal memory pressure of 17% and low spatial locality, the performance of pure S-COMA is 6.7 times worse than CC-NUMA's at memory pressures as low as 30%. Although the performance of both R-NUMA and VC-NUMA are significantly more stable than that of S-COMA, they too suffer from thrashing by the time memory pressure reaches 70%. The source of this performance degradation is the same as in em3d and barnes - increasing kernel overhead and (to a lesser degree) induced cold misses. Once again, R-NUMA induces fewer remote accesses than CC-NUMA [4], but the kernel overhead required to support page relocation is such that R-NUMA underperforms CC-NUMA by 75% at 70% memory pressure and by almost a factor of two at 90% memory pressure. Once again, VC-NUMA's backoff algorithm proves to be more effective than R-NUMA's, but it still underperforms CC-NUMA by roughly 40% at high memory pressures. AS-COMA, on the other hand, deposits a reasonable subset of "hot" pages into the page cache and then backs off from replacing further pages once it detects thrashing. As a result, even for a program with almost no spatial locality, AS-COMA is able to converge to CC-NUMA-like performance (or better) across all memory pressures. At 90% memory pressure, AS-COMA outperforms VC-NUMA by 35% and R-NUMA by 90% at high memory pressures, and it remains within 5% of CC-NUMA's performance. The slight degradation compared to CC-NUMA is due to the short period of thrashing that occurs before AS-COMA can detect it and completely stop relocations.

[2]Note that barnes is very compute-intensive, and a problem size that can be simulated in a reasonable amount of time requires only approximately 100 home pages per node of data. Since there are only about 50 free pages per node available for page replication at 70% memory pressure, we did not simulate barnes at higher memory pressures since the results would be heavily skewed by small sample size effects.

Applications in the second category (fft, ocean, and lu) exhibit good page-grained locality. All three applications only have a small set of "hot" pages, which can be easily replicated using a small page cache, or references to remote pages are so localized that the small (128-byte) RAC in our simulation was able to satisfy a high percentage of remote accesses [4]. As a result, thrashing never occurs and the various backoff schemes are not invoked. Thus, the performance of the three hybrid algorithms is almost identical.

In summary, for applications that do not suffer frequent remote cache misses or for which the active working set of remote pages is small at any given time, all of the hybrid architectures perform quite well, often outperforming CC-NUMA. However, for applications with less spatial locality or larger working sets, the more aggressive remapping backoff mechanism used by AS-COMA is crucial to achieving good performance. In such applications, AS-COMA outperformed the other hybrid architectures by 20% to 90%, and either outperformed or broke even with CC-NUMA even at extreme memory pressures. Given programmers' desire to run the largest problem size that they can on their machines, this stability of AS-COMA at high memory pressures could prove to be an important factor in getting hybrid architectures adopted.

6. Conclusions

The performance of hardware distributed shared memory is governed by three factors: (i) remote memory latency, (ii) the number of remote misses, and (iii) the software overhead of managing the memory hierarchy. In this paper, we evaluated the performance of five DSM architectures (CC-NUMA, S-COMA, R-NUMA, VC-NUMA, and AS-COMA) with special attention to the third factor, system software overhead. Furthermore, since users of SMPs tend to run the largest applications possible on their hardware, we paid special attention to how well each architecture performed under high *memory pressure*.

We found that at low memory pressure, architectures that were most aggressive about mapping remote pages into the local page cache (S-COMA and AS-COMA) performed best. In our study, S-COMA and AS-COMA outperformed the other architectures by up to 17% at low memory pressures. As memory pressure increased, however, it became increasingly important to reduce the rate at which remote pages were remapped into the local page cache. S-COMA's performance usually dropped dramatically at high memory pressures due to thrashing. The performance of VC-NUMA and R-NUMA also dropped at high memory pressures, albeit not as severely as S-COMA. This thrashing phenomenom has been largely ignored in previous studies, but we found that it had a significant impact on performance, especially at the high memory pressures likely to be preferred by power users.

In contrast, AS-COMA's software-based scheme to detect thrashing and reduce the rate of page remappings caused it to outperform VC-NUMA and R-NUMA by up to 90% at high memory pressures. AS-COMA is able to fully utilize even a small page cache by mapping a subset of "hot" pages locally, and then backing off further remapping. This mechanism caused AS-COMA to outperform even CC-NUMA in five out of the six applications we studied, and only underperform CC-NUMA by 5% in the sixth.

Consequently, we believe that hybrid CC-NUMA/S-COMA architectures can be made to perform effectively at all ranges of memory pressures. At low memory pressures, aggressive use of available DRAM can eliminate most remote conflict misses. At high memory pressures, reducing the rate of page remappings and keeping only a subset of "hot" pages in the small local page cache can lead to performance close to or better than CC-NUMA. To achieve this level of performance, the overhead of system software must be carefully considered, and careful attention must given to avoiding needless system overhead. AS-COMA achieves these goals.

7 Acknowledgements

We would like to thank Leigh Stoller who supported the simulator environment, the anonymous reviewers, and the members of *Avalanche* project at the University of Utah for their helpful comments on drafts of this paper.

References

[1] W. Bolosky, R. Fitzgerald, and M. Scott. Simple but effective techniques for NUMA memory management. In *Proceedings of the 12th ACM Symposium on Operating Systems Principles*, pages 19–31, Dec. 1989.

[2] S. Chandra, J. Larus, and A. Rogers. Where is time spent in message-passing and shared-memory programs? In *Proceedings of the 6th Symposium on Architectural Support for Programming Languages and Operating Systems*, pages 61–73, Oct. 1994.

[3] 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.

[4] C.-C. 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.

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

[6] 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.

[7] 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.

[8] M. Mckusick, K. Bostic, M. Karels, and J. Quarterman. *The Design and Implementation of the 4.4BSD operating system*, chapter 5 Memory Management, pages 117–190. Addison-Wesley Publishing Company Inc, 1996.

[9] 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.

[10] 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.

[11] L. Stoller, R. Kuramkote, and M. Swanson. PAINT- PA instruction set interpreter. Technical Report UUCS-96-009, University of Utah - Computer Science Department, Sept. 1996.

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

[13] B. Verghese, S. Devine, A. Gupta, and M. Rosenblum. Operating system support for improving data locality on CC-NUMA compute servers. In *Proceedings of the 7th Symposium on Architectural Support for Programming Languages and Operating Systems*, Oct. 1996.

[14] 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.