Appears in *Proceedings of the 24th Annual International Symposium on Computer Architecture*

Reactive NUMA: A Design for Unifying S-COMA and CC-NUMA

Babak Falsafi and David A. Wood
Computer Sciences Department
University of Wisconsin, Madison
1210 W. Dayton Street
Madison, WI 53706
{babak,david}@cs.wisc.edu

Abstract

This paper proposes and evaluates a new approach to directory-based cache coherence protocols called *Reactive NUMA* (R-NUMA). An R-NUMA system combines a conventional CC-NUMA coherence protocol with a more-recent Simple-COMA (S-COMA) protocol. What makes R-NUMA novel is the way it dynamically reacts to program and system behavior to switch between CC-NUMA and S-COMA and exploit the best aspects of both protocols. This reactive behavior allows each node in an R-NUMA system to independently choose the best protocol for a particular page, thus providing much greater performance stability than either CC-NUMA or S-COMA alone. Our evaluation is both qualitative and quantitative. We first show the theoretical result that R-NUMA's worst-case performance is bounded within a small constant factor (i.e., two to three times) of the best of CC-NUMA and S-COMA. We then use detailed execution-driven simulation to show that, in practice, R-NUMA usually performs better than either a pure CC-NUMA or pure S-COMA protocol, and no more than 57% worse than the best of CC-NUMA and S-COMA, for our benchmarks and base system assumptions.

1 Introduction

Clusters of symmetric multiprocessors, or SMPs, have emerged as a promising approach to building large-scale shared-memory parallel machines [15,16]. The relatively high volumes of these small-scale parallel servers make them extremely cost-effective as building blocks. By connecting these low-cost nodes, system designers hope to construct large-scale parallel machines with better cost-performance than has been previously possible [3].

This work is supported in part by Wright Laboratory Avionics Directorate, Air Force Material Command, USAF, under grant #F33615-94-1-1525 and ARPA order no. B550, NSF PYI Award CCR-9157366, NSF Grants MIP-9225097 and MIP-9625558, an IBM graduate fellowship, and donations from A.T.&T. Bell Laboratories, Hewlett Packard, IBM Corporation, and Sun Microsystems. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Wright Laboratory Avionics Directorate or the U.S. Government.

To preserve software compatibility, these clusters use a directory-based cache coherence protocol to support a shared-memory abstraction despite having memory physically distributed across the nodes. Such systems have previously employed either a cache-coherent non-uniform memory access (CC-NUMA) protocol [15,16], a Simple-COMA (S-COMA) protocol [11], or provided support for both [22,8]. In a CC-NUMA system, remote data may be cached in a CPU's cache or a per-node cluster cache. References not satisfied by these hardware caches must be sent to the referenced page's home node to obtain the requested data (and enforce any necessary coherence actions). An S-COMA system uses the exact same coherence protocol, but allocates part of the local node's main memory to act as a large cache for remote pages. S-COMA is much cheaper and simpler to implement than earlier COMA systems [10,12] because it uses standard address translation hardware as "tags" for the page cache. Because S-COMA requires only incrementally more hardware than CC-NUMA, some systems have proposed providing support for both protocols [22,8].

A potential disadvantage of DSM clusters is the relatively large ratio of remote to local miss times. For example, the Sequent STiNG's remote misses are roughly ten times slower than local misses [16]. Conversely, in a full-integrated implementation like the recently-announced SGI Origin2000 [1], the ratio can be as small as two to three times. Thus an application's performance on a DSM cluster will be very sensitive to the frequency of remote misses.

S-COMA can potentially perform much better than CC-NUMA on these machines. This is because the S-COMA page cache is part of main memory, and can be much larger than a CC-NUMA cluster cache. Thus, S-COMA potentially results in significantly fewer remote misses, and achieves better performance. CC-NUMA machines can reduce this difference by allocating pages on the same node as the processor that uses them [17], dynamically migrating pages [4,24], and replicating read-only pages [24]. These techniques have been shown to dramatically reduce the number of remote misses for some applications. Unfortunately, they provide no help for read-write shared data, which are quite frequent in other applications. For example, Verghese, et al., found that 90% of user data misses in a commercial relational database application were to read-write shared pages [24]. S-COMA permits replication of these read-write pages, thus eliminating unnecessary remote references.

Conversely, S-COMA is not without its costs. Allocating space in the S-COMA page cache incurs substantial overhead, in the form of operating system intervention to set up the local translation. The page cache must eliminate many misses to amortize this initial overhead (and the subsequent replacement overhead). This is further complicated by the large—i.e., page—granularity. Programs with large sparse data sets may see severe internal fragmentation and thrash the S-COMA page cache. Thus CC-NUMA may have

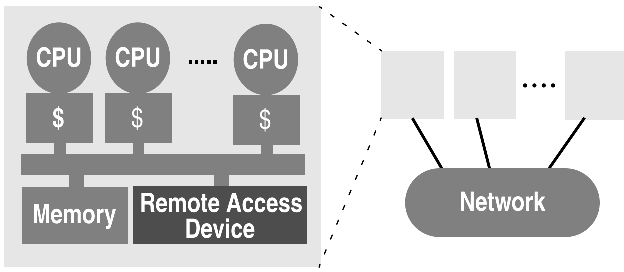

FIGURE 1. A distributed shared-memory machine.

substantially better performance for some applications and reference patterns.

In this paper, we propose a Reactive NUMA (R-NUMA) protocol that dynamically reacts to program and system behavior to switch between CC-NUMA and S-COMA protocols. This reactive behavior provides much greater performance stability than either CC-NUMA or S-COMA alone, by allowing each node in an R-NUMA system to independently choose the best protocol for a particular page. The algorithm initially allocates all remote pages as CC-NUMA, but maintains a per-node, per-page count of the number of times a block is refetched as a result of a conflict or capacity miss. When the refetch count exceeds a threshold, the operating system intervenes and reallocates the page in the S-COMA page cache. R-NUMA requires very little additional hardware—primarily a set of per-node, per-page counters—beyond what is needed to support both CC-NUMA and S-COMA.

We evaluate the protocol both qualitatively and quantitatively. We first present a simple analytical model indicating that R-NUMA performs competitively with respect to CC-NUMA and S-COMA. This result guarantees that R-NUMA never performs much worse than the best of either S-COMA or CC-NUMA. We then use detailed execution-driven simulation to show that, in practice, R-NUMA's reactive behavior often leads to better performance than either a pure CC-NUMA or pure S-COMA protocol. Quantitative results also confirm our theoretical result: R-NUMA performs no more than 57% worse than the best of the two underlying protocols. Conversely, for one application CC-NUMA was 179% slower than S-COMA; for another S-COMA was 315% slower than CC-NUMA. Thus, for our benchmarks and system assumptions, R-NUMA often achieves the best performance, and is never much worse than the best possible. This superior performance stability makes R-NUMA a very attractive alternative for future shared-memory machines.

The next section describes the basic distributed shared-memory machine structure that we study in this paper, and provides more details of the CC-NUMA and S-COMA protocols. Section 3 presents the Reactive NUMA protocol and its qualitative analysis. Section 4 and Section 5 describe the simulation methodology we use to quantitatively evaluate the protocols and the results, respectively. Finally, Section 6 summarizes the results and conclusions in this paper.

2 DSM Hardware Support

Figure 1 illustrates the basic distributed shared-memory machine organization that we study in this paper. Each node is a symmetric multiprocessor (SMP) workstation with four processors connected via a coherent bus to an interleaved memory. A *Remote Access*

(a)

(b)

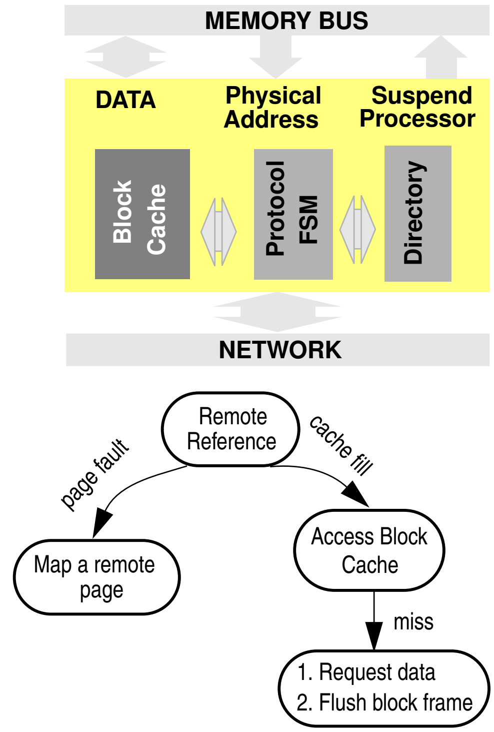

FIGURE 2. Caching remote data in CC-NUMA: (a) anatomy of a CC-NUMA RAD, (b) flow of a remote miss.

Device (RAD) implements a directory-based cache coherence protocol to extend the shared-memory abstraction across the nodes. This device implements the same basic coherence protocol in all systems; CC-NUMA, S-COMA, and R-NUMA simply differ in where remote data is cached.

All systems implement a global physical address space, where the high-order bits encode the node id. CC-NUMA systems reference global addresses directly; S-COMA uses a simple SRAM mapping table to translate local physical addresses to global physical addresses. R-NUMA does both, since it supports both protocols. All systems run a single operating system image, but maintain separate per-node page tables to permit independent allocation decisions and reduce TLB fill latency and contention.

2.1 CC-NUMA

Most previous distributed shared-memory machines have been CC-NUMA machines [15,2,16,5,18]. Figure 2(a) illustrates the CC-NUMA RAD we consider in this paper. Like the Sequent STiNG [16], our CC-NUMA RAD is equipped with a remote cluster cache for maintaining recently referenced remote data blocks. This cache acts as another level in the node's cache hierarchy, but unlike the processor caches, only holds remote data. In the remainder of this paper, we refer to this CC-NUMA cache as a *block cache* to differentiate it from S-COMA's page-granularity cache. A hardware protocol controller manages accesses to the block cache and directory, services messages from remote nodes, and requests remote data on the behalf of the node.

Figure 2 (b) illustrates the flow of events on a remote reference in CC-NUMA. The first processor to access a remote page within each node results in a (soft) page fault. The operating system's page fault handler maps the page to a CC-NUMA global physical address, updating the node's page table. Subsequent references to the same page by any processor within the SMP node result in cache block fills on the memory bus. The CC-NUMA RAD snoops for global physical addresses on the bus and satisfies the cache fill requests if the data resides in the block cache. Upon a miss in the block cache, the RAD allocates a block frame, writing back a dirty block if necessary, and sends a request for the data to the remote node.

The block cache improves CC-NUMA's performance by reducing the number of remote misses. Block caches can either be small, fast, and built with SRAM, or large, slow, and built with DRAM. The latter reduces the number of remote misses, but at the expense of increasing the remote miss latency and controller occupancy. To keep latencies and occupancies comparable between CC-NUMA and S-COMA we consider only relatively small, fast block caches in this paper.

Because the block cache is still relatively small, CC-NUMA's performance is very sensitive to data allocation and placement. LaRowe, et al. [14] have shown that a good initial allocation works well for many scientific applications. We use a first-touch migration policy which is both simple and has been shown to substantially eliminate unnecessary traffic [17]. In this policy, a user-invoked directive on every node initiates page migration at the start of the parallel phase of the program. Upon the first request for each page, the home node migrates the page to the requester, assuming the first requester is likely to prove a frequent requester. This is especially true for some regular scientific applications that specifically "touch" pages to ensure their proper placement [26]. Dynamic replication/migration can further improve performance by replicating code and read-only data pages, and migrating pages that are mostly accessed by a single processor [24]. Unfortunately, these techniques fail for data pages that are actively shared among multiple processors.

2.2 S-COMA

Cache-Only Memory Architectures (COMA) allow remote data to reside in both the node's cache hierarchy and main memory. For example, the SICS DDM [10] and Kendall Square Research KSR1 [19] replicate and migrate data at cache block granularities among the node's memories. Because data is allocated at subpage granularities, these machines require the use of significant additional hardware to determine the physical location of a datum.

Simple COMA (S-COMA) [11,21] greatly simplifies the original COMA approach by decoupling data allocation and naming from coherence. Remote data is allocated and mapped at page granularity using standard virtual address translation hardware, much like page-based DSMs [7]. A region of main memory is set aside as a *page cache* for remote data pages. An S-COMA remote access device (RAD), Figure 3 (a), maintains coherence using the same basic coherence protocol as the CC-NUMA RAD. The essential extra hardware is a set of fine-grain access control tags—two bits per block to detect when the RAD must inhibit memory and intervene—and an auxiliary SRAM translation table with one entry per page—to convert between local physical addresses (i.e., the page cache) and global physical addresses (i.e., the home address). Because operating system software handles the more complex operations of allocation and migration, S-COMA is much simpler than "full" COMA implementations.

Figure 3 (b) illustrates the S-COMA algorithm. On a node's first reference to a remote page, a page fault occurs and is handled by

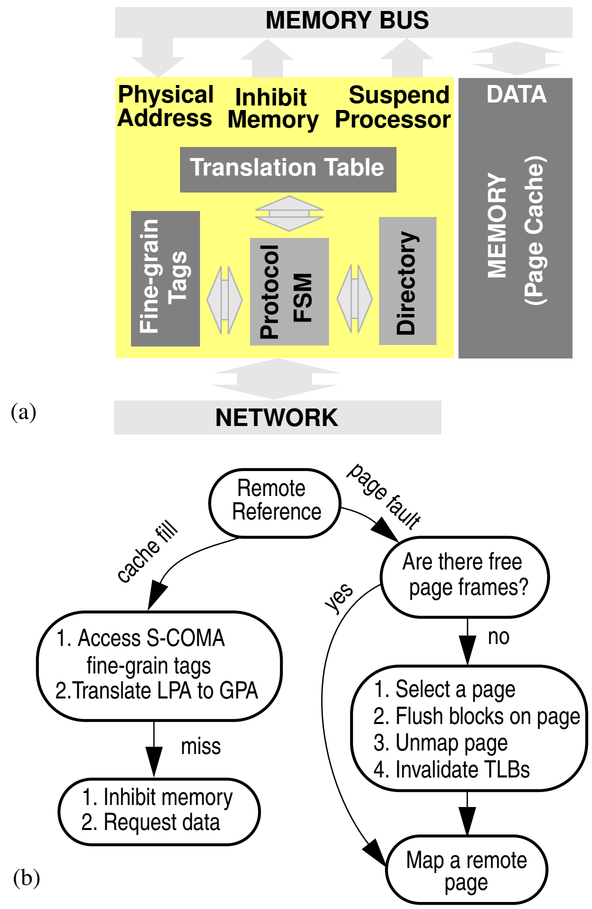

FIGURE 3. Caching remote data in S-COMA: (a) anatomy of an S-COMA RAD, (b) flow of a remote miss.

the operating system. If there are no free page frames available in the page cache, the operating system selects a victim, unmaps and flushes all dirty blocks back to home node, and shoots down the TLBs on the *local* node. When a page frame is available, the operating system initializes the page table, the RAD's auxiliary translation table, and fine-grain access control tags. Subsequent references find the page mapped, and hits and misses are detected by the fine-grain access control tags. Hits are serviced by local memory. Misses are detected by the RAD, which inhibits memory, translates to the corresponding global physical address, and communicates with the home node to obtain the requesting block.

S-COMA can potentially outperform CC-NUMA because it can exploit the node's large main memory to cache remote data. S-COMA can dynamically tailor the fraction of memory used to cache remote data, in response to an application's needs. S-COMA's remote cache is also fully-associative, because S-COMA uses the standard virtual address translation hardware to locate remote pages.

S-COMA's simplicity and fully-associative page cache come at a cost, however. Many parallel applications do not have sufficient spatial locality to fully utilize remote pages, leading to internal fragmentation. Irregular applications and regular applications with large strides are particularly susceptible to this problem. These applications may incur significant internal fragmentation, and require a prohibitively large page cache to fit their remote working sets. If the page cache is not sufficiently large, the result is frequent S-COMA page replacement (i.e., thrashing) and a rapid decrease in performance.

3 Reactive NUMA

Reactive NUMA (R-NUMA) tries to combine the best aspects of both CC-NUMA and S-COMA. R-NUMA is based on the observation that remote data pages can be classified into two groups. *Reuse* pages contain data structures that are accessed many times within the same node, and thus exhibit substantial remote traffic due to capacity and conflict misses in the node's cache hierarchy. Conversely, *communication* pages are used primarily to exchange data between nodes, and thus mainly exhibit coherence misses. R-NUMA attempts to distinguish between these two types of pages, and store reuse pages in an S-COMA-like page cache, while limiting communication pages to the node's cache hierarchy and CC-NUMA-like block cache. R-NUMA dynamically detects when communication pages become reuse pages, and vice versa.

3.1 Mechanisms for R-NUMA

An R-NUMA machine must provide mechanisms for caching remote data pages both as CC-NUMA and S-COMA pages. The operating system maps CC-NUMA pages directly to a remote global physical address and S-COMA pages to a local physical address in the page cache. The R-NUMA RAD snoops for both CC-NUMA global physical addresses and (local) physical addresses in the S-COMA page cache. Note that R-NUMA need not require any additional hardware. For example, s3.mp [18] and the Stanford FLASH [13] already provide sufficient mechanisms to implement both CC-NUMA and S-COMA. Because they use programmable controllers, implementing R-NUMA should just be a software change.

Figure 4 (a) illustrates that an R-NUMA RAD looks like a combination of an S-COMA RAD and a CC-NUMA RAD. A block cache serves as a backup device for CC-NUMA-mapped pages, a set of S-COMA fine-grain tags provide access control for S-COMA-mapped pages, a simple SRAM mapping table translates local S-COMA physical addresses to the corresponding global physical addresses, and a directory maintains coherence information for each page for which a node is the designated home.

The R-NUMA protocol separates data allocation from coherence. Coherence is a *global* operation, where all nodes cooperate to enforce the shared-memory abstraction. Data allocation is a *local* decision, which determines whether a particular cache block (page) should be replicated in the S-COMA page cache or CC-NUMA block cache. This separation allows each node to independently decide whether to manage a particular shared page as CC-NUMA or S-COMA.

R-NUMA requires a mechanism to decide when to switch between CC-NUMA and S-COMA. The key difference between reuse and communication pages is the number of remote capacity and conflict misses a page incurs in the block cache. When a block gets evicted from the block cache due to its limited capacity or a set conflict, the next subsequent reference to that block will miss, causing a *refetch* from the home node. The directory can detect refetches by simply keeping track of when a node requests a block that the directory state indicates it already has. Assuming a non-notifying protocol, this is trivial for read-only blocks. Handling

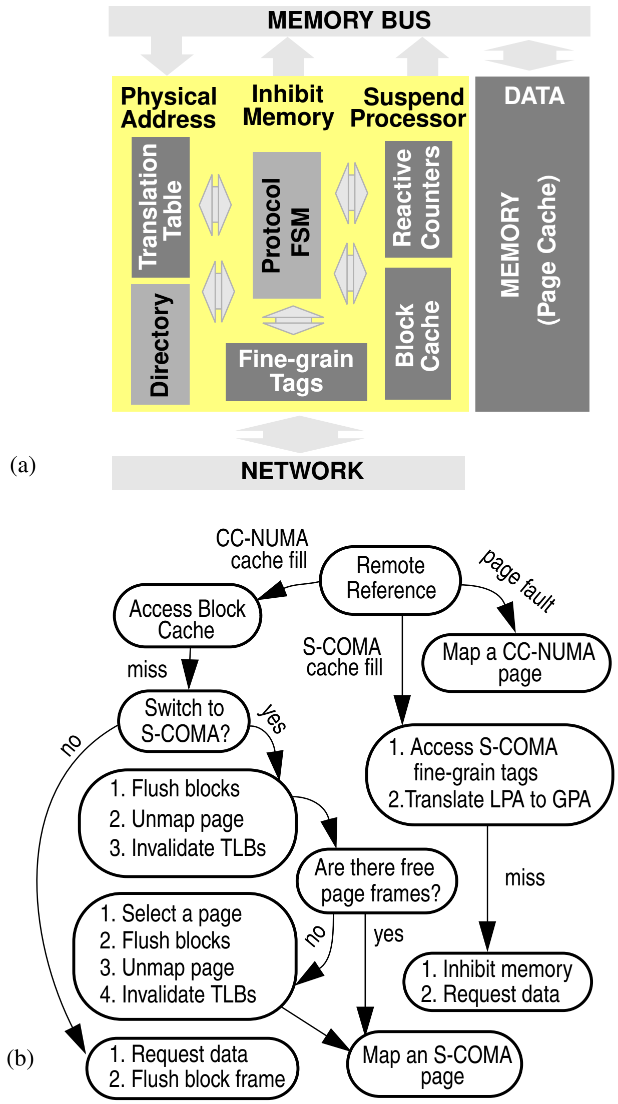

FIGURE 4. Caching remote data in R-NUMA: (a) anatomy of an R-NUMA RAD, (b) flow of a remote miss.

read-write blocks generally requires adding an additional state to indicate that a processor previously held an exclusive block, but voluntarily wrote it back. The system then keeps track of the number of refetches on a per-node, per-page basis. The directory can either maintain the counts itself, similar to the page migration counts in the SGI Origin2000 [1], or communicate the information back to the requester on the reply message. We assume that each R-NUMA RAD maintains a set of per-page counters for its node and generates an interrupt when the count exceeds a preset threshold.

Figure 4 (b) illustrates the flow of events for caching remote data in R-NUMA. The first reference to an unmapped page results in a page fault. The operating system initially maps the page CC-NUMA. Further references are handled by the R-NUMA RAD, either supplying them from the block cache or fetching them from the home node. The RAD uses the per-page counters to detect when the number of refetches exceeds the threshold. When this occurs, the operating system is invoked to relocate the page from CC-NUMA to S-COMA. Relocation requires unmapping the CC-NUMA page, flushing all locally cached blocks from that page, and allocating and mapping a new S-COMA page (possibly cleaning a page, if no free ones exist).

3.2 Qualitative Performance

R-NUMA's performance depends upon how well it selects the appropriate mapping for a given page. If an off-line oracle selects the mapping, then the R-NUMA protocol always performs at least as well as either CC-NUMA or S-COMA, since it combines the hardware resources of both. Unfortunately, real systems must rely on on-line algorithms.

Instead, we present a simple intuitive model to analyze the worst-case behavior of R-NUMA. Our model shows that an application's execution time under R-NUMA is always within a small constant factor of the execution time under the best of CC-NUMA and S-COMA.

In the interest of brevity and clarity, we make several simplifying assumptions about the behavior of the system. We compare performance relative to an "ideal" CC-NUMA machine with an infinite capacity block cache. The finite capacities of the actual block and page caches result in extra overheads due to refetches in the block cache, replacements in the page cache, and relocation of pages between the block and page caches. We limit our analysis to these extra overheads for a single remote page. The results generalize to multiple pages because they hold for worst-case reference patterns.

Assume a CC-NUMA with a block cache of a given size, an S-COMA with a page cache of a given size, and an R-NUMA with block and page caches equal in size to their counterparts in the CC-NUMA and S-COMA. Also assume that the cost of fetching a block from a remote node is much higher than a local memory access. Our results remain qualitatively valid for systems that violate these assumptions, however, a further discussion is beyond the scope of this paper.

Parameter	Description
$C_{refetch}$	Cost of refetching a remote block
$C_{allocate}$	Cost of allocating/replacing a page
$C_{relocate}$	Cost of relocating a page
T	Relocation threshold value
$O_{CC\text{-}NUMA}$	Per-page overhead of CC-NUMA
$O_{S\text{-}COMA}$	Per-page overhead of S-COMA
$O_{R\text{-}NUMA}$	Per-page overhead of R-NUMA

TABLE 1. Parameters for the performance model.

Table 1 depicts the parameters we use in our performance model. $C_{refetch}$ is the cost of refetching a block from a remote node; $C_{allocate}$ is the cost of allocating and later replacing a page; $C_{relocate}$ is the cost of relocating a page from CC-NUMA to S-COMA; T is the number of refetches before R-NUMA relocates a CC-NUMA page to an S-COMA page. Our model uses these parameters to compute $O_{CC\text{-}NUMA}$, $O_{S\text{-}COMA}$ and $O_{R\text{-}NUMA}$, which represent the additional per-page overheads of the three machines as compared to our ideal machine respectively.

R-NUMA's per-page behavior digresses from CC-NUMA's when a page incurs more refetches than specified by the threshold T. R-NUMA relocates such a page to the page cache in order to convert remote block fetches to local memory accesses. R-NUMA performs worst when a page relocates from the block cache to the page cache and is not referenced again before being replaced. In this case, CC-NUMA's overhead ($O_{CC\text{-}NUMA}$) is only $TC_{refetch}$ whereas R-NUMA would incur additional overheads of relocating the blocks on the CC-NUMA page, and allocating and subsequently replacing an S-COMA page for a total of $O_{R\text{-}NUMA} = TC_{refetch} + C_{relocate} + C_{allocate}$. Therefore, R-NUMA's performance is worse than CC-NUMA by at most

$$\frac{O_{R\text{-}NUMA}}{O_{CC\text{-}NUMA}} = \frac{TC_{refetch} + C_{relocate} + C_{allocate}}{TC_{refetch}}. \qquad \text{(EQ 1)}$$

R-NUMA's worst-case performance with respect to S-COMA also occurs for the same case. S-COMA's per-page overhead ($O_{S\text{-}COMA}$) would be simply $C_{allocate}$ whereas R-NUMA's overhead ($O_{R\text{-}NUMA}$) would include the additional overheads of refetching T blocks and relocating a page, i.e., $TC_{refetch} + C_{relocate} + C_{allocate}$. Hence, S-COMA will outperform R-NUMA by at most

$$\frac{O_{R\text{-}NUMA}}{O_{S\text{-}COMA}} = \frac{TC_{refetch} + C_{relocate} + C_{allocate}}{C_{allocate}}. \qquad \text{(EQ 2)}$$

Our goal is to minimize the worst-case performance of R-NUMA with respect to both CC-NUMA and S-COMA. The right hand sides of the above two equations are intersecting functions of T. At the point of intersection, R-NUMA's relative worst-case performance is equal to

$$\frac{O_{R\text{-}NUMA}}{O_{CC\text{-}NUMA}} = \frac{O_{R\text{-}NUMA}}{O_{S\text{-}COMA}} = 2 + \frac{C_{relocate}}{C_{allocate}} \qquad \text{(EQ 3)}$$

at the threshold value of $T = \dfrac{C_{allocate}}{C_{refetch}}$.

Equation 3 indicates that the bound on worst-case performance of R-NUMA depends on the cost of relocation relative to the cost of page allocation/replacement. Relocation includes generating an interrupt when the number of refetches reaches the threshold, and moving the blocks from the block cache to the page cache. In a high-performance implementation with support for fast interrupts and mechanisms for (locally) relocating blocks, $C_{relocate}$ will be small compared to $C_{allocate}$, and the worst-case performance bound will be close to 2. In a less aggressive implementation, the cost of relocation will be dominated by interrupt overhead and flushing the blocks from the block cache, $C_{relocate}$ will be approximately equal to $C_{allocate}$, and the worst-case performance will be close to 3.

Equation 3 also indicates that the threshold value at the point of intersection simply depends on the per-page overheads of CC-NUMA and S-COMA. As such, the value is a function of the cost of page allocation/replacement and the cost of a remote block fetch, and is independent of the cost of relocation.

In practical terms, the worst-case performance analysis proves that R-NUMA performs no more than three times worse than either a vanilla CC-NUMA or S-COMA system. In fact, since block refetch, page allocation/replacement, and page relocation over-

Application	Problem	Input Data Set
barnes	Barnes-Hut N-body simulation [26]	16K particles
cholesky	Blocked sparse Cholesky factorization [26]	tk16.O
em3d	3-D electromagnetic wave propagation [9]	76800 nodes, 15% remote, 5 iters
fft	Complex 1-D radix-$\sqrt{n}$ six-step FFT [26]	64K points
fmm	Fast Multipole N-body simulation [26]	16K particles
lu	Blocked dense LU factorization [26]	512x512 matrix, 16x16 blocks
moldyn	Molecular dynamics simulation [6]	2048 particles, 15 iters
ocean	Ocean simulation [26]	258x258 ocean
radix	Integer radix sort [26]	1M integers, radix 1024
raytrace	3-D scene rendering using ray-tracing [26]	car

TABLE 3. Applications and input parameters.

heads are only a few out of many components of execution time, the practical "bound" is much less than three. However, while bounding worst-case performance is important for some applications, e.g., on-line transaction processing and process control, most users focus on average performance. Moreover, the threshold value that maximizes R-NUMA's performance in practice may be different from the one that minimizes worst-case performance. In the remainder of this paper, we show that R-NUMA's ability to dynamically allocate some pages to the S-COMA page cache and others to the CC-NUMA block cache can significantly improve performance. We also show that R-NUMA performs well even with a much smaller block cache than CC-NUMA. Finally, we present results on the sensitivity of R-NUMA's performance on relocation threshold value and overhead.

4 Methodology

To compare practical implementations of R-NUMA, CC-NUMA, and S-COMA, we simulate a distributed shared-memory machine consisting of a network of eight SMP nodes (Figure 1). Each node is a 4-way multiprocessor with 400 MHz dual-issue statically scheduled processors—modeled after the Ross HyperSparc—interconnected by a 100 MHz split-transaction bus. A snoopy MOESI coherence protocol—modeled after Sparc's MBus protocol—keeps the caches within each node consistent. We assume perfect instruction caches[1] but model data caches and their contention at the memory bus accurately. We further assume a point-to-point network with a constant latency of 100 cycles but model contention at the network interfaces.

Our block cache is a writeback direct-mapped SRAM cache. The cache maintains inclusion—with respect to the node's cache hierarchy—for remote blocks cached in the read-write state but not for those cached in the read-only state. Cache inclusion for read-write blocks greatly simplifies the interaction between the DSM CC-NUMA protocol and the commodity workstation MOESI protocol. Maintaining inclusion for the remote blocks in the read-only state would require a very large block cache. Instead we opted for not maintaining inclusion for read-only blocks. However, since MBus does not implement cache-to-cache transfer for blocks that are not owned by a processor, read requests to read-only remote blocks that miss in the block cache are forwarded to the home node even if there are copies of the block in other processor caches on the node.

Operation	Cost (processor cycles)
block operations	
SRAM access	8
DRAM access	56
local cache fill	69
remote fetch	376
page operations	
soft traps	2000
TLB shootdown	200
allocation/replacement or relocation	3000~11500

TABLE 2. Base line system assumptions.

Both S-COMA and R-NUMA implement a simple *Least Recently Missed* page replacement policy. This policy is similar to classical LRU, but the page frame list is re-ordered only on remote misses rather than on each reference. This can be approximated in practice by maintaining per-page hardware miss counters which the operating system periodically samples. However, since page replacement policies are beyond the scope of this paper, our model of S-COMA hardware simply maintains the necessary information and communicates it to the operating system at the time of a page fault.

Table 2 presents the costs of block and page operations in processor cycles for our base system assumptions. SRAM devices include the block cache, S-COMA fine-grain tags and translation table, and R-NUMA reactive counters. DRAM accesses correspond to accesses to the page cache. Soft traps include page faults and R-NUMA relocation interrupts. Page allocation/replacement involves taking a soft trap, invalidating the (local) TLBs, and flushing the blocks back to the home node. The overhead varies depend-

1. The scientific codes we study have low instruction cache miss ratios. This assumption may not hold for all applications.

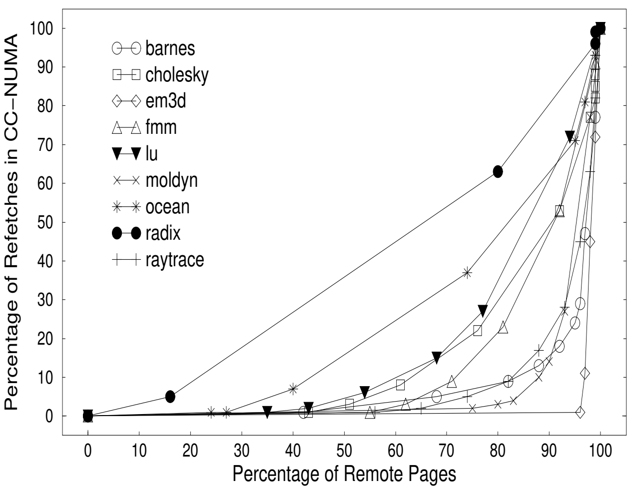

FIGURE 5. Characterizing remote pages.

The figure plots the cumulative distribution of refetches as a function of the fraction of remote pages in a CC-NUMA machine with a 32-Kbyte block cache. The figure omits *fft,* because it fits in the node's cache hierarchy and incurs no capacity or conflict misses.

ing on the number of blocks flushed. Page relocation uses similar mechanisms as page allocation/replacement and incurs the same overheads.

Table 3 presents the applications we use in this study and the corresponding input parameters. *Barnes*, *cholesky*, *fft*, *fmm*, *lu*, *ocean*, *radix* and *raytrace* are from the SPLASH-2 [26] benchmark suite. *Em3d* is a shared-memory implementation of the Split-C benchmark [9]. Moldyn is a shared-memory implementation of a CHARMM-like [6] molecular dynamics application.

Our application data set sizes are selected to be small enough so as to not require prohibitive simulation cycles, while being large enough to maintain the intrinsic communication and computation characteristics of the parallel application. Woo, et al., characterize the behavior of SPLASH-2 applications in terms of working sets and show that for most of the applications, the data sets provided have a primary working set that fits in an 8-Kbyte cache [26]. We, therefore, assume 8-Kbyte (direct-mapped) processor caches to compensate for the small size of the data sets.

In this study, our base system assumes a CC-NUMA block cache equal in size to the sum of all the processor cache sizes. This assumption helps mitigate any adverse effects due to the inclusion requirement of read-write blocks. Consequently, a four-processor node will have a 32-Kbyte CC-NUMA block cache. To compensate for the lower cost of DRAM as compared to SRAM, our base system assumes an S-COMA page cache of 320 Kbytes, a factor of 10 larger than our CC-NUMA block cache. We further assume a much smaller 128-byte block cache in R-NUMA than that in CC-NUMA. We present results in Section 5 that indicate that R-NUMA performs well even with such a small block cache.

5 Results

In this section, we present results from our simulation experiments. We first motivate the results by presenting a characterization of remote pages in a CC-NUMA machine. Next, we present numbers comparing the performance of CC-NUMA, S-COMA, and R-NUMA. In the rest of the section, we study the sensitivity of R-NUMA's performance to block cache size, relocation threshold value and overhead.

Application	CC-NUMA	R-NUMA	
	RW pages	refetches	replacements
barnes	97%	21%	2%
cholesky	28%	30%	15%
em3d	100%	0%	0%
fmm	99%	142%	2%
lu	82%	21%	70%
moldyn	98%	0%	0%
ocean	96%	36%	4%
radix	15%	125%	1%
raytrace	5%	41%	5%

TABLE 4. Characterizing block refetches and page replacements in CC-NUMA and R-NUMA.

The table presents the fraction of block refetches due to pages that incur both read and write coherence misses in CC-NUMA, and block refetches and page replacements in R-NUMA as a percentage of those in CC-NUMA and S-COMA. The table omits *fft* because it incurs no capacity or conflict misses in CC-NUMA and only a small number of page replacements in S-COMA. The numbers correspond to a CC-NUMA with a 32-Kbyte block cache, an S-COMA with 320-Kbyte page cache, and an R-NUMA with 128-byte block cache and a 320-Kbyte page cache and threshold value of 64.

5.1 Characterizing Pages in CC-NUMA

R-NUMA offers a performance advantage over CC-NUMA if an application incurs a large number of capacity and conflict misses on remote data in a CC-NUMA machine. R-NUMA also outperforms S-COMA when the majority of the capacity and conflict misses in CC-NUMA are due to a small fraction of remote pages that can fit in the page cache. Conversely, R-NUMA can perform worse, if the page cache is too small to accommodate the set of remote pages that account for most of the capacity and conflict misses in CC-NUMA. Therefore, R-NUMA's performance relative to CC-NUMA and S-COMA depends on the fraction of reuse and communication remote pages in the application.

Figure 5 illustrates the fraction of remote pages that are responsible for a given percentage of block refetches in CC-NUMA—due to capacity and conflict misses. The graphs indicate that in four of the applications, less than 10% of the remote pages account for over 80% of the capacity and conflict misses in CC-NUMA. With the exception of *radix*, an additional 20% of remote pages (for a total of 30%) account for just under 70% of the refetches in all of the applications. *Radix* performs an all-to-all communication, where processors march through a large number of remote pages writing to small number of blocks. As such, the remote pages for the most part exhibit similar behavior, and the capacity and conflict misses are evenly distributed among the pages.

Table 4 (second column from left) presents the fraction of block refetches in CC-NUMA due to pages that incur both read and write sharing traffic. The table indicates that with the exception of some of the kernels and *raytrace*, read-write remote pages account for over 80% of the block refetches in all of the applications. This result indicates that our first-touch migration is very effective in eliminating block refetch traffic due to pages that are not shared—i.e., pages that are always used by one node but initially allocated on another node. Moreover, simple replication of (read-only) remote pages which has been shown to be effective in mulitiprogammed workloads [24], will not help pages with read-write traffic. *Raytrace*, is the only application that proves to be a good candidate for replication schemes because most of the remote pages contain read-only data structures.

Not surprisingly, two of the kernels (*cholesky* and *radix*) also exhibit a large fraction of remote pages with read-only traffic. The kernels are representative of common computations that are typically found in larger applications. As such, much of the data structures that appear as read-only remote data in the kernel, are in fact the results of other intermediate stages of computation. Depending on the read-write sharing behavior of the data throughout the application, dynamic read-only replication schemes may or may not be effective. R-NUMA's relocation overheads are relatively small—only referenced blocks are replicated—and as such R-NUMA can help reduce read-only traffic even in small kernel computations.

5.2 Base System Results

R-NUMA's performance depends on the relative frequency of block refetches in the R-NUMA and CC-NUMA block caches, as well as page replacements in the R-NUMA and S-COMA page caches. Table 4 also presents the number of block refetches and page replacements in R-NUMA as a fraction of those in CC-NUMA and S-COMA. The numbers compare a CC-NUMA with a 32-Kbyte block cache, an S-COMA with a 320-Kbyte page cache, and an R-NUMA with a 128-byte block cache, a 320-Kbyte page cache, and a relocation threshold value of 64.

The table indicates that R-NUMA substantially reduces the block refetch traffic relative to CC-NUMA in all but two of the applications. R-NUMA also virtually eliminates the page replacement traffic relative to S-COMA in most of the applications. R-NUMA increases block refetches in *fmm* and *radix* relative to CC-NUMA because of its small block cache. Moreover, R-NUMA's page cache is too small to accommodate all the reuse pages, causing the pages to bounce between the block and page caches. The combined effect of a small block cache and page cache too small to contain all the reuse pages increases the overall number of refetches in these applications.

Figure 6 compares the performance of CC-NUMA, S-COMA, and R-NUMA. The graphs present the execution times on a CC-NUMA with a 32-Kbyte block cache, an S-COMA with a 320-Kbyte page cache, and an R-NUMA with 128-byte block cache, a 320-Kbyte page cache and a relocation threshold value of 64. The graphs are normalized to a CC-NUMA with an infinite block cache—i.e., one in which the block cache is large enough to hold all of the remote data.

Not surprisingly, the performance of CC-NUMA and S-COMA varies across applications. Applications whose sharing occurs in a small number of localized regions fit well in the node's cache hierarchy and the block cache. In contrast, a page cache favors applications with dense sharing patterns, because the overhead of allocation/replacement of a page can be amortized over a large number of blocks per page. Dense sharing patterns also result in lower page fragmentation, and thus place less pressure on the page cache.

The graphs corroborate our intuition (from Section 3.2) that R-NUMA either outperforms or is competitive with CC-NUMA and S-COMA. In the best case, R-NUMA reduces execution time by

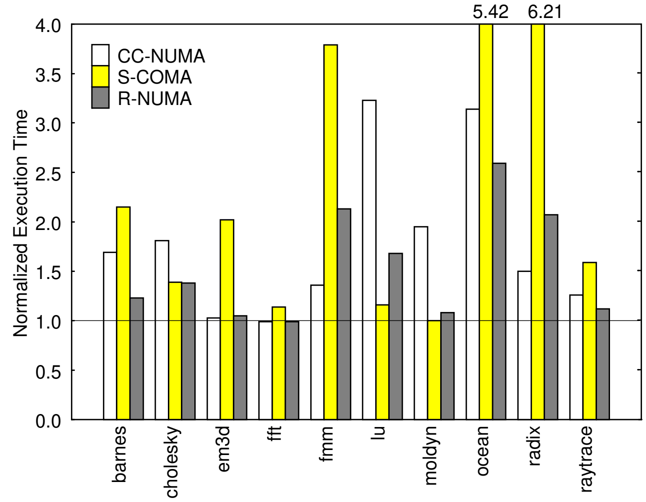

FIGURE 6. Comparing performance of CC-NUMA, S-COMA and R-NUMA.

The figure plots execution times on a CC-NUMA with a 32-Kbyte block cache, S-COMA with 320-Kbyte page cache, and R-NUMA with a 128-byte block cache, a 320-Kbyte page cache, and a relocation threshold value of 64. The numbers are normalized to a CC-NUMA with an infinite block cache, i.e., a block cache that can hold all of the referenced remote data.

37% in the best of CC-NUMA and S-COMA. In the worst case, R-NUMA increases execution time by only 57% in the best of the two protocols. In comparison, CC-NUMA performs as much as 179% worse than S-COMA, and S-COMA performs as much as 315% worse than CC-NUMA. Thus, R-NUMA exhibits much less sensitivity to a particular application's behavior and provides superior performance stability over either CC-NUMA or S-COMA alone.

We now examine the individual applications in more detail. In *em3d* and *fft* communication is of producer-consumer nature, where remote pages primarily exchange recently produced data. These applications incur minimal number of capacity and conflict misses in the 32-Kbyte block cache and hence perform well in CC-NUMA. S-COMA fails to provide enough page frames to hold all the remote pages for either of these applications, thus resulting in lower performance. Much like CC-NUMA, R-NUMA's block cache provides enough (temporary) storage for the remote data, and thus R-NUMA performs as well as CC-NUMA.

Moldyn performs well in S-COMA because the page cache can capture the complete set of remote pages in this application. In *moldyn*, a small number of reuse pages account for most of the capacity and conflict misses in CC-NUMA. R-NUMA simply relocates these pages into the page cache and performs much like S-COMA.

Cholesky, *fmm, lu*, *ocean* and *radix* are examples of applications in which a large fraction of remote pages are responsible for the capacity and conflict misses in CC-NUMA's block cache. S-COMA's (R-NUMA's) page cache is large enough to accommodate a large fraction of the remote pages in *cholesky* and *lu*. R-NUMA reduces most of the refetches in these applications and therefore outperforms CC-NUMA. R-NUMA also reduces most of the page replacements for *cholesky.* However, because of load-imbalance in *lu* [26], page replacements occur more frequently on the critical path, so the decrease is not as great as might be expected. Likewise, a small fraction of *lu*'s remote pages bounce between R-NUMA's block cache and page cache. Page replacements incur larger overheads in R-NUMA than in S-COMA because a page must reach the threshold value before relocation. Nevertheless, R-NUMA improves performance up to a factor of two over CC-NUMA and stays competitive with S-COMA in these applications.

Remote data in *fmm*, *ocean* and *radix* are too large to fit in the page cache. S-COMA's performance suffers for these applications because of frequent replacements in the page cache. In *fmm* and *radix,* CC-NUMA improves performance over S-COMA by up to a factor of 4 because the remote working sets of these applications fit in the 32-Kbyte block cache. R-NUMA eliminates much of the page replacements in S-COMA (Table 4), but increases the number of refetches in CC-NUMA because the 320-Kbyte page cache cannot contain the large number of reuse pages in these applications. R-NUMA remains competitive with CC-NUMA, increasing execution time by at most 57% in these applications. *Ocean* exhibits a large remote working set which does not even fit in CC-NUMA's block cache. Although R-NUMA outperforms both CC-NUMA and S-COMA for this application, block and page traffic remain high.

R-NUMA performs best when an application exhibits a small number of reuse remote pages that frequently miss in CC-NUMA's block cache, but the application's overall set of remote pages is too large to fit in S-COMA's page cache. R-NUMA can detect and relocate the reuse pages into the page cache, thereby eliminating much of the capacity and conflict misses in the block cache. This is the case for *barnes* and *raytrace*. In these applications, R-NUMA virtually eliminates all of the refetches and replacements in CC-NUMA and S-COMA and outperforms both.

5.3 Cache Size Sensitivity

CC-NUMA's performance lies in its ability to cache the (remote) working set of data in the node's processors caches and the block cache. Many classes of applications exhibit large temporal localities and small primary working set sizes [26,20]. Small CC-NUMA caches typically result in high performance because they are adequate to hold the primary working set of these applications.

Unfortunately, there are classes of applications that exhibit poor temporal locality and large primary working sets—e.g., commercial databases [16]. CC-NUMA's inability to cache these application's working sets severely degrades performance. R-NUMA can mitigate this problem by allowing portions of the working set with small temporal but large spatial localities to relocate to a large page cache. R-NUMA's performance, like S-COMA's, lies in its ability to cache these application's large working sets. In this section we study the sensitivity of CC-NUMA's and R-NUMA's performance to cache sizes.

Figure 7 plots CC-NUMA and R-NUMA execution times normalized to a system with an infinite block cache. We present CC-NUMA numbers for a small 1-Kbyte block cache and a 32-Kbyte block cache large enough to hold the primary working sets of most of the applications. R-NUMA numbers correspond to our base system assumptions of a 128-byte block cache with a 320-Kbyte page cache, a system with a larger 32-Kbyte block cache and a 320-Kbyte page cache, and a system with a 128-byte block cache and 40-Mbyte page cache, which is large enough to hold the working sets of all of the applications.

The graphs indicate that the applications can be grouped into three categories based on their working set sizes of reuse pages. *Em3d* and *fft* are examples of applications with small reuse working sets. In these applications, communication primarily consists of

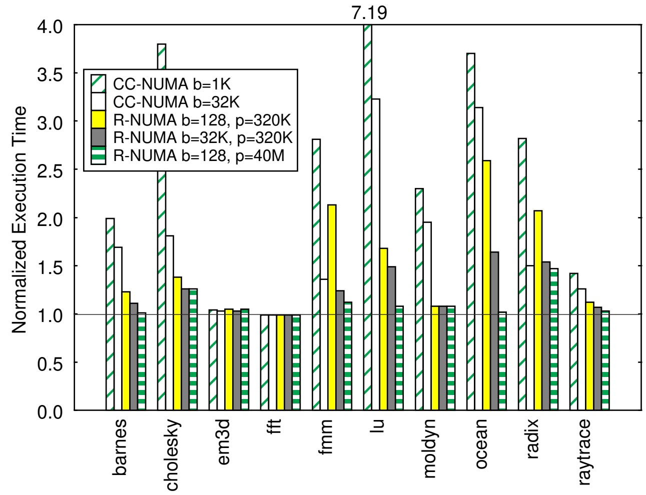

FIGURE 7. Performance sensitivity of CC-NUMA and R-NUMA to cache size.

The figure compares the performance sensitivity of CC-NUMA and R-NUMA to cache sizes. The figures plot execution times normalized to a CC-NUMA with an infinite block cache. CC-NUMA numbers correspond to a 1-Kbyte block cache (b=1K) and a 32-Kbyte block cache (b=32K). R-NUMA numbers correspond to a 128-byte block cache with a 320-Kbyte page cache (b=128, p=320K), a 32-Kbyte block cache with a 320-Kbyte page cache (b=32K, p=320K), and a 128-byte block cache with a 40-Mbyte page cache (b=128, p=40M). R-NUMA uses a relocation threshold value of 64.

exchanging data between a producer and a consumer. The graphs indicate that these applications achieve high performance even with a 1-Kbyte block cache. Similarly, *barnes*, *moldyn,* and *raytrace* all have primary reuse working sets that fit in a very small block cache but require a much larger (> 32 Kbytes) cache to capture the complete set of remote data. R-NUMA achieves high performance in these applications even with a small (128-byte) block cache by moving the large portions of the reuse working sets into the page cache.

In the second category are those applications whose primary reuse working sets do not fit in a small 1-Kbyte cache, but do fit in a larger 32-Kbyte cache. *Cholesky*, *fmm,* and *radix* fall in this category. A small block cache severely impacts CC-NUMA's performance in these applications and increases execution time by up to a factor of 2. R-NUMA exhibits performance sensitivity to block cache size only when the reuse working set does not fit in the page cache. *Fmm* and *radix* have large and sparse working sets which result in fragmentation in the page cache. R-NUMA's performance improves up to 90% with either a large 32-Kbyte block cache or a large 40-Mbyte page cache. R-NUMA manages to capture *cholesky*'s reuse working set in a 320-Kbyte page cache, and hence shows no sensitivity to block cache size.

Lu and *ocean* comprise the third category of applications. In these applications, the primary reuse working set does not fit even in the larger 32-Kbyte block cache. In these applications, CC-NUMA's performance exhibits very high sensitivity to block cache size; execution times in CC-NUMA increase by up to a factor of 7 compared to a machine with an infinite size block cache. Much as in the second category of applications, R-NUMA's performance becomes sensitive to block cache size for applications whose reuse working set does not fit in the page cache, as in *ocean*.

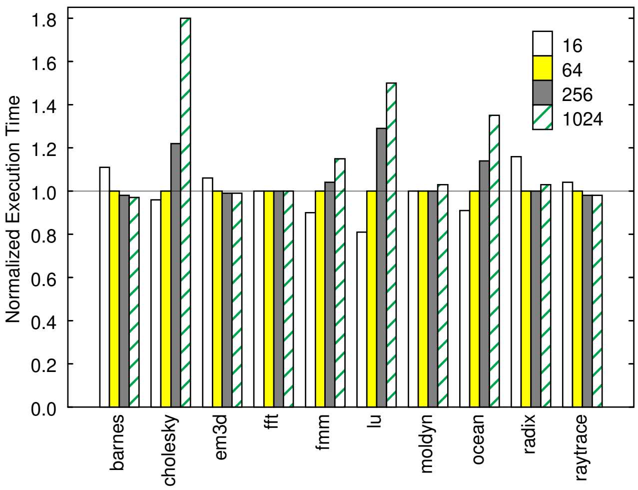

FIGURE 8. Performance sensitivity of R-NUMA to relocation threshold value.

The figure plots R-NUMA's performance sensitivity to relocation threshold value. R-NUMA relocates a page from the block cache into the page cache when it incurs as many capacity and conflict misses as specified by the threshold value. The numbers correspond to execution times on an R-NUMA with a 128-byte block cache and a 320-Kbyte page cache. The numbers are normalized to an R-NUMA with a relocation threshold value of 64.

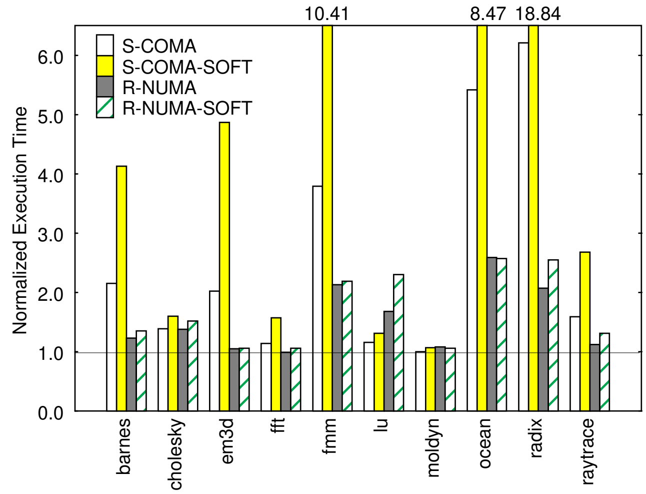

FIGURE 9. Performance sensitivity of S-COMA and R-NUMA to page-fault and TLB invalidation overheads.

The figure compares the sensitivity of S-COMA's and R-NUMA's performance to TLB invalidation and page fault overheads. Our base systems, S-COMA and R-NUMA, assume page fault handling times of 5 µs, and TLB hardware invalidation times of 0.5 µs. The slower systems, S-COMA-SOFT and R-NUMA-SOFT, assume a higher page fault handling time of 10 µs, and TLB software invalidation times of 5 µs. The page caches are of size 320 Kbytes. R-NUMA uses a 128-byte block cache and a relocation threshold value of 64. The numbers are normalized to execution times on a CC-NUMA with an infinite block cache.

5.4 Threshold Sensitivity

Worst-case performance analysis (Section 3.2) dictates that the relocation threshold be determined so as to bound the performance of the worst-case reference stream by a small constant. However, real applications rarely exhibit such worst-case behavior, so the threshold that gives the best performance in practice may be different. We would like to select a relocation threshold value which is low enough to allow reuse pages to relocate as quickly as possible, while high enough to prevent pages with low capacity and conflict miss rates from relocation. Performance sensitivity of R-NUMA to the threshold value is therefore directly related to the fraction of reuse pages in the remote working set of an application; a large fraction of reuse pages can benefit from low threshold values and vice versa.

Figure 8 plots R-NUMA's performance for various relocation threshold values. The R-NUMA configuration is a 128-byte block cache and 320-Kbyte page cache. The numbers are normalized to execution times on an R-NUMA with threshold value of 64—i.e., a page is selected for relocation when it incurs 64 capacity or conflict misses in the block cache. The graphs indicate that in all but three of the applications, R-NUMA's performance varies by at most 27%.

The figure also corroborates our intuition that a larger fraction of reuse pages in an application favor a smaller threshold value. *Cholesky*, *fmm*, *lu* and *ocean* all exhibit a large fraction of remote pages with high capacity and conflict miss rates in CC-NUMA (Figure 5). A threshold value of 16 can improve performance by up to 25% over a threshold value of 64 in these applications.

5.5 Relocation Overhead Sensitivity

Another factor that has a first order effect on the performance of a reactive protocol is the page relocation overhead: a larger overhead requires a higher threshold to keep R-NUMA competitive with CC-NUMA and S-COMA. Relocation of remote data from the block cache to the page cache involves taking a page fault (when the number capacity and conflict misses on a page reaches the threshold), allocating/replacing a page frame, and relocating the blocks to it. Page fault handling times can vary depending on the implementation and can take thousands of cycles on many commodity workstations [23]. Replacement requires invalidating the TLBs on a node, which may involve (slow) inter-processor interrupts if the processors are not equipped with the necessary hardware support—such as TLB invalidate transactions on the memory bus [25]. In this section we study the sensitivity of R-NUMA's performance to page fault and TLB invalidation overheads.

Figure 9 compares the sensitivity of S-COMA's and R-NUMA's performance to page fault and TLB invalidation overheads. S-COMA and R-NUMA correspond to our base case assumptions of 5 µs for page fault handling and 0.5 µs for (fast) TLB hardware invalidation. S-COMA-SOFT and R-NUMA-SOFT correspond to our slower systems with a 10 µs page fault handling time and a much higher 5 µs for a TLB software invalidation using inter-processor interrupts. The per-page allocation/replacement and relocation overheads are therefore approximately 3 times higher in the slower systems. The page caches are all of size 320 Kbytes, and R-NUMA's block cache is 128 bytes.

The graphs indicate that the performance of S-COMA is highly sensitive to page fault and TLB invalidation overheads. This is not surprising, because applications whose remote working sets are larger than the page cache exhibit high page replacement rates. In these cases, an increase in replacement overhead directly impacts performance; the execution time in more than half of the applications increases by up to a factor of 3 with a 3-fold increase in per-page relocation overhead.

R-NUMA's performance, however, is much less sensitive to page fault and TLB invalidation overheads. The execution time in R-NUMA-SOFT increases by at most 25% for all of but one of our applications. Because R-NUMA substantially reduces the replacement rate in the page cache (Table 4), its performance is not as sensitive to an increase in overhead.

Lu is our only application whose execution time on R-NUMA-SOFT increases by 40%. *Lu*'s working set of remote data primarily consist of reuse pages. Working set sizes significantly vary across nodes because of load-imbalance inherent to the blocking algorithm in this application for small data sets [26]; in *lu,* two nodes are responsible for more than 50% of the page replacements in the system. Because these slow nodes are on the critical path of the execution, an increase in the relocation overhead directly impacts execution time.

6 Conclusions

In this paper, we proposed and evaluated R-NUMA, a design for combining a conventional CC-NUMA with a more recent Simple-COMA in a single distributed shared-memory machine. R-NUMA's novelty lies in its ability to dynamically react to program and system behavior and select between CC-NUMA and S-COMA to exploit the best remote caching strategy on a per-page basis. CC-NUMA capitalizes on short-term temporal locality and small-scale spatial locality. Conversely, S-COMA's large page cache can exploit longer-term temporal localities, but only for applications with large-granularity spatial localities. R-NUMA monitors per-page miss behavior to determine when a page should be switched to the other protocol. R-NUMA's reactive behavior provides much greater performance stability than either CC-NUMA or S-COMA alone.

We presented a simple qualitative result that R-NUMA's worst-case performance is bounded within a small constant factor of the best of CC-NUMA and S-COMA. We then presented quantitative results based on execution-driven simulation of a distributed shared-memory system. Our results indicated that in practice: (i) R-NUMA usually outperforms or performs as well as the best of either CC-NUMA or S-COMA, (ii) when R-NUMA performs worse than the best protocol, the performance gap is much smaller than that predicted by our qualitative analysis, and (iii) R-NUMA never performs worse than both CC-NUMA and S-COMA. For the shared-memory applications we studied, R-NUMA was either best or within a few percent of best for seven of the ten; for the others it performed worst than the best protocol by at most 57%. In comparison CC-NUMA's and S-COMA's performance differed by as much as 315%.

We also investigated the sensitivity of R-NUMA's performance to cache sizes, relocation threshold, and relocation overhead. The results indicated that R-NUMA performs well with a very small block cache unless the working set of reuse pages is too large to fit in the page cache, which was the case in three of the applications we studied. In comparison, CC-NUMA exhibited very high sensitivity in seven of the applications when the primary working set was too large to fit in the block cache. R-NUMA did not exhibit significant sensitivity to relocation threshold or relocation—i.e., page allocation/replacement—overhead. In contrast, S-COMA's performance in half of the applications was highly sensitive to an increase in page allocation/replacement overhead.

The quantitative results we presented in this paper are closely tied to the application workload and system characteristics we studied. Many of the applications were extensively tuned to take advantage of locality in small caches. The relative performance of a reactive system may vary with both application (e.g., working set size) and system (e.g., cache sizes) characteristics. Our qualitative result on the reactive system's competitiveness, however, holds across a wide range of applications and systems.

Acknowledgements

We would like to thank Steve Reinhardt for helping with the development of our simulator, Beng-Hong Lim and Sandra Irani for their comments on our performance models, and Scott Breach, Erik Hagersten, Mark Hill, Andreas Moshovos, Jon Wade, and Bob Zack for their comments on earlier drafts of this paper.

References

[1] Silicon Graphics Origin Technology. http://www.sgi.com/Products/hardware/servers/technology/index.html.

[2] Anant Agarwal, Ricardo Bianchini, David Chaiken, Kirk L. Johnson, David Kranz, John Kubiatowicz, Beng-Hong Lim, Kenneth Mackenzie, and Donald Yeung. The MIT Alewife Machine: Architecture and Performance. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pages 2–13, June 1995.

[3] Tom Anderson, David Culler, and David Patterson. A Case for NOW (Networks of Workstations). *IEEE Micro*, 15(1):54–64, February 1995.

[4] David Black, Anoop Gupta, and Wolf-Dietrich Weber. Competitive management of distributed shared memory. In *Proceedings of COMPCON*, March 1989.

[5] Tony Brewer. A Highly Scalable System Utilizing up to 128 PA-RISC Processors. http://www.convex.com/tech_cache/ps/SPP_Arch.times.ps.

[6] B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, D . J. States, S. Swamintathan, and M. Karplus. Charmm: A program for macromolecular energy, minimization, and dynamics calculation. *Journal of Computational Chemistry*, 4(187), 1983.

[7] John B. Carter, John K. Bennett, and Willy Zwaenepoel. Implementation and Performance of Munin. In *Proceedings of the 13th ACM Symposium on Operating System Principles (SOSP)*, pages 152–164, October 1991.

[8] John B. Carter, Al Davis, Ravindra Kuramkote, Chei-Chi Kuo, Leigh B. Stoller, and Mark Swanson. Avalanche: A Communication and Memory Architecture for Scalable Parallel Computing. In *Workshop on Scalable Shared-Memory Multiprocessors*, 1995. http://www.cs.utah.edu:80/projects/avalanche/.

[9] D. E. Culler, A. Dusseau, S. C. Goldstein, A. Krishnamurthy, S. Lumetta, T. von Eicken, and K. Yelick. Parallel Programming in Split-C. In *Proceedings of Supercomputing '93*, pages 262–273, November 1993.

[10] Erik Hagersten, Anders Landin, and Seif Haridi. DDM–A Cache-Only Memory Architecture. *IEEE Computer*, 25(9):44–54, September 1992.

[11] Erik Hagersten, Ashley Saulsbury, and Anders Landin. Simple COMA Node Implementations. In *Proceedings of the 27th Hawaii International Conference on System Sciences*, January 1994.

[12] Kendall Square Research. Kendall Square Research Technical

Summary, 1992.

[13] Jeffrey Kuskin et al. The Stanford FLASH Multiprocessor. In *Proceedings of the 21st Annual International Symposium on Computer Architecture*, pages 302–313, April 1994.

[14] Rick LaRowe and Carla Ellis. Experimental Comparison of Memory Management Policies for NUMA multiprcessors. *ACM Transactions on Computer Systems*, 9(4):319–363, November 1991.

[15] Daniel Lenoski, James Laudon, Kourosh Gharachorloo, Wolf-Dietrich Weber, Anoop Gupta, John Hennessy, Mark Horowitz, and Monica Lam. The Stanford DASH Multiprocessor. *IEEE Computer*, 25(3):63–79, March 1992.

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

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

[18] A. Nowatzyk, M. Monger, M. Parkin, E. Kelly, M. Borwne, G. Aybay, and D. Lee. S3.mp: A Multiprocessor in a Matchbox. In *Proc. PASA*, 1993.

[19] E. Rosti, E. Smirni, T.D. Wagner, A.W. Apon, and L.W. Dowdy. The KSR1: Experimentation and Modeling of Poststore. In *Proceedings of the 1993 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 74–85, May 1993.

[20] Edward Rothberg, Jaswinder Pal Singh, and Anoop Gupta. Working Sets, Cache Sizes, and Node Granularity Issues for Large-Scale Multiprocessors. In *Proceedings of the 20th Annual International Symposium on Computer Architecture*, pages 14–25, June 1993.

[21] Ashley Saulasbury, Tim Wilkinson, John Carter, and Anders Landin. An Argument for Simple COMA. In *Proceedings of the First IEEE Symposium on High-Performance Computer Architecture*, pages 276–285, January 1995.

[22] Ashley Saulsbury and Andreas Nowatzyk. Simple COMA on S3.MP. http://playground.Sun.COM/pub/S3.mp/simple-coma/isca-95/present.html.

[23] Chandramohan A. Thekkath and Henry M. Levy. Hardware and Software Support for Efficient Exception Handling. In *Proceedings of the Sixth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 110–119, San Jose, California, 1994.

[24] Ben Verghese, Scott Devine, Anoop Gupta, and Mendel Rosenblum. Operating System Support for Improving Data Locality on CC-NUMA Compute Servers. In *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS VII)*, October 1996.

[25] Shlomo Weiss and James E. Smith. *Power and PowerPC*. Morgan Kaufmann Publishers, Inc., 1994.

[26] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pages 24–36, July 1995.