Exploiting Coarse Grain Non-Shared Regions in Snoopy Coherent Multiprocessors

Andreas Moshovos
moshovos@eecg.toronto.edu
Computer Engineering Group Technical Report
Electrical and Computer Engineering
University of Toronto
December 2, 2003

Abstract

It has been shown that many requests miss in all remote nodes in shared memory multiprocessors. We are motivated by the observation that this behavior extends to much coarser grain areas of memory. We define a region to be a continuous, aligned memory area whose size is a power of two and observe that many requests find that no other node caches a block in the same region even for regions as large as 16K bytes (it has already been known that this phenomenon applies to the special cases of a block or a page). We propose RegionScout, a family of simple filter mechanisms that dynamically detect most non-shared regions. A node with a RegionScout filter can determine in advance that a request will miss in all remote nodes. RegionScout filters are implemented as a layered extension over existing snoop-based coherence systems. They require no changes to existing coherence protocols or caches and impose no constraints on what can be cached simultaneously. Their operation is completely transparent to software and the operating system. RegionScout filters require little additional storage and a single additional global signal. These characteristics are made possible by utilizing imprecise information about the regions cached in each node. Since they rely on dynamically collected information RegionScout filters can adapt to changing sharing patterns. We present two applications of RegionScout: In the first RegionScout is used to avoid broadcasts for non-shared regions thus reducing bandwidth. In the second RegionScout is used to avoid snoop induced tag lookups thus reducing energy.

1 Introduction

There are workload, technology and cost trends that make shared memory multiprocessors an increasingly popular architecture [12,26]. Today there are applications (e.g., databases, file and mail servers, multimedia/entertainment and communications) with sufficient parallelism that shared memory multiprocessors can exploit. In addition, the increasing levels of chip integration make single chip/module multiprocessors viable and an attractive alternative for utilizing additional on-chip resources [4,11,24]. Finally, the increased cost and complexity of building modern processors make using multiple identical cores a cost-effective option for high integration devices.

With the increased popularity of shared memory multiprocessors comes renewed interest in techniques for improving their efficiency and performance, particularly in techniques related to coherence. This is exemplified by the recent advances in coherence speculation (e.g., [13,14,16,17,18,21,22]) and in power-aware coherence [9,20,23]. The same cost and technology trends that favor using multiple identical cores also favor techniques that are as little intrusive as possible at the hardware and software levels. At the same time, the abundance of on-chip resources creates opportunities for novel coherence implementations [4].

We propose RegionScout a technique that exploits coarse grain sharing patterns in snoop-based shared memory multiprocessors with potential applications in reducing bandwidth, latency and energy. Previous work has shown that many requests miss in all remote nodes in shared memory multiprocessors. We observe that many shared memory requests not only do not find a matching block in any remote node but also they do not find a block in the same region (where a region is a continuous, aligned memory area whose size is a power of two).

RegionScout comprises a family of filters that dynamically observe coarse grain sharing and allow nodes to detect in advance that a request will miss in all remote nodes. With RegionScout when a node sends a request in addition to

block-level sharing information it receives region-level sharing information. If a region is identified as not shared subsequent requests from the same node that fall in the same region are identified as non-shared without having to probe any other node. Normal coherence activity allows the detection of regions that become shared preserving correctness. RegionScout filters utilize imprecise information about the regions that are cached in each node in the form of a hashed bit vector [5]. For this reason, they detect many but not all requests that would miss in all other nodes. This loss in coverage allows RegionScout to be completely transparent to software and avoids imposing artificial constraints on what can be cached simultaneously. RegionScout filters require no changes to the underlying coherence mechanisms: when possible they simply return a "not shared" response without having to consult the existing coherence protocol. Their implementation is flexible allowing a trade-off between filter storage cost and accuracy. To demonstrate the utility of RegionScout filters we investigate two applications. We show that RegionScout filters can be used to avoid broadcasts for some requests. Avoiding broadcasts frees bandwidth which may be used more fruitfully. We also show that RegionScout filters can be used to avoid snoop-induced tag lookups thus reducing energy in the higher level of the local memory hierarchy.

The rest of this paper is organized as follows: In section 2 we detail our simulation environment and methodology. In section 3 we motivate RegionScout filters by showing that many requests do not find a block in the same region in remote nodes for various shared memory multiprocessor configurations. We comment on related work in section 4. We present the principle of operation for RegionScout filters in section 5. We also discuss their implementation and additional benefits that come "for free". In section 6, we show experimentally that RegionScout filters are effective and can reduce bandwidth and energy. Finally, we summarize this work in section 7.

2 Methodology

Our simulator is based on Simplescalar v2.0 [6] with Manjikian's shared memory extensions [15]. We made extensive changes to the simulator and the shared memory support libraries since Manjikian's model is limited to direct-mapped caches, uses pseudo system calls for synchronization and does not implement all PARMACS macros necessary to run all the SPLASH2 benchmarks. For synchronization we modelled the *load linked* and *store conditional* MIPS instructions which we used to implement all synchronization primitives including MCS locks [19]. We added support for set-associative caches, shared or private L2 caches and a bus model. We used a modified version of GNU's gcc v2.7.2 to produce optimized PISA binaries for the SPLASH2 benchmarks [25] shown in table 1. We do not include Water as a result of a math library bug.

Table 1: SPLASH2 applications and input parameters.

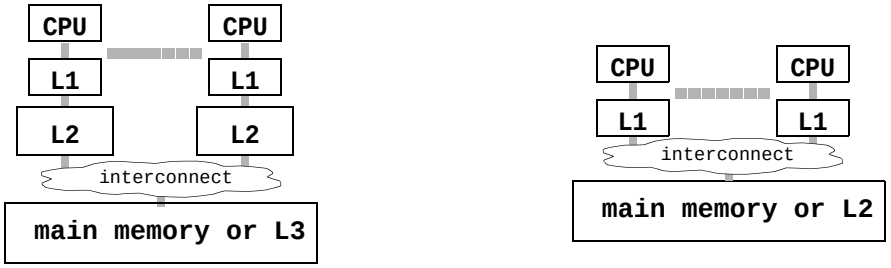
Benchmark	Input Parameters	Benchmark	Input Parameters
barnes	*16k particles*	***ocean (contig.)***	*258x258 grid w/ defaults*
cholesky	*tl29.O*	***radiosity***	*-batch -room*
fft	*256k complex data points*	***radix***	*2M keys*
fmm	*16k particles*	***raytrace***	*balls4.env*
lu (contig.)	*512x512 matrix, B=16*	***volrend***	*256x256x126 voxels*

We use the two models of shared multiprocessor systems shown in figure 1. Under model *LocalL2* each node has private L1 and L2 caches and there is a shared L3 or main memory. This is representative of conventional SMPs. Under model *SharedL2* nodes have private L1 caches and there is a shared L2. This is representative of simple CMPs [11] and of processors like Power4 [24] that are popular as building blocks for larger SMP systems. We include both models since energy- and bandwidth-wise they behave differently (with the differences being primarily a function of block and cache sizes). We use a MESI coherence protocol.

We modelled systems with two, four or eight processors. Unless otherwise noted, LocalL2 systems have L2 caches of the specified size that are eight-way set-associative with 64-byte blocks and 32Kbyte L1 caches that are four-way set-associative with 32-byte blocks. In SharedL2 systems the L1 caches use 32-byte blocks and are four-way set-associative and of the specified size.

3 Motivation

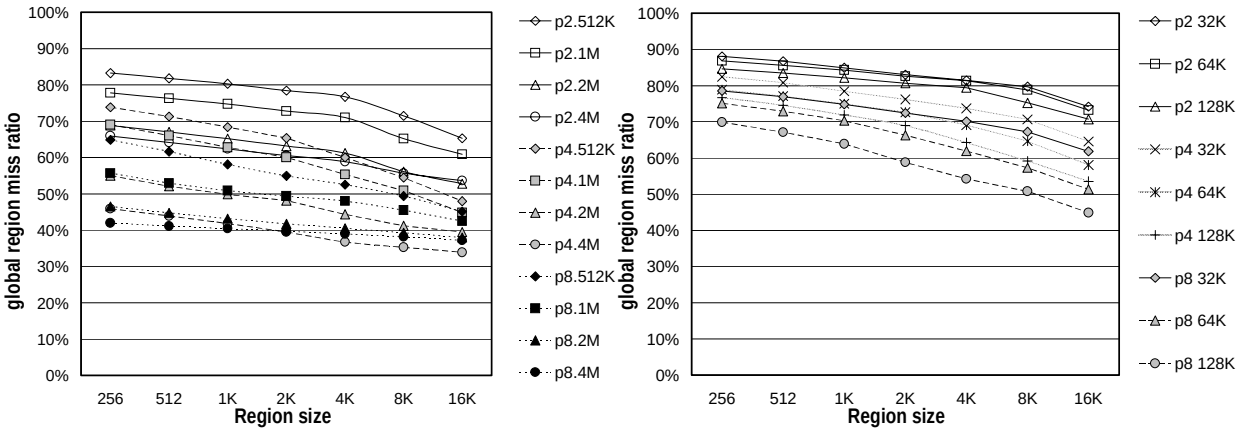
For clarity, let us first define the terms *region, region miss* and *global region miss*. A *region* is an aligned, continuous section of memory whose size is a power of two. A request for a block B in a cache C results in a *region miss* if C holds no block in the same region as B including B itself. We were motivated by the observation that often memory requests result in region misses in *all* remote caches, or, in a *global region miss*. This is a generalization of observations made by earlier work in snoop energy reduction. Specifically, earlier work has shown that this property holds



(a) LocalL2: Local L1 and L2, shared L3 **(b) SharedL2:** Local L1, shared L2

Figure 1: Two shared memory multiprocessor models. (a) **LocalL2**: local L1 and L2 caches and a shared L3. (b) **SharedL2**: local L1 caches and a shared L2. The interconnect is functionally a bus but may be implemented via other components [8].

for the special cases where the region is a block [20] or a page [9]. Intuitively, one would expect that this property would apply to other region sizes as well. This intuition holds true as shown in figure 2 where we measure the global region miss ratio for systems with different number of nodes and cache sizes. The *global region miss ratio* is the fraction of all coherent memory requests that result in a global region miss. We consider regions of 256 bytes up to 16K bytes and systems with two, four or eight processors. In part (a) the per node L2 cache size is varied from 512K bytes to four megabytes. In part (b) the per node L1 cache size is varied from 32Kbytes to 128Kbytes.



(a) LocalL2 **(b)** SharedL2

Figure 2: Average global region miss ratio for various shared memory multiprocessor systems (different curves) and region sizes (Y-axis). (a) Systems with local L1 and L2 caches and shared L3. (b) Systems with local L1 caches and a shared L2. Labels are *pN.S* where *N* is the number of nodes and *S* is the L2 and L1 cache size for parts (a) and (b) respectively.

In general, the average global region miss ratio is inversely proportional to private cache sizes, node count and region size. (An anomaly is observed in (a) when the number of nodes increases from four to eight. This is primarily a result of our barrier implementation and the memory allocation of related structures.) Even if we consider the worst case (four node LocalL2 with four Mbyte L2 caches) more than one in three requests results in a region miss for a region as large as 16K bytes. For typical organizations the fraction of global region misses is much higher. For example, for the four node LocalL2 system with 512K L2 caches, the average global region miss ratio varies from 74% down to 48% depending on the region size. For the SharedL2 four node system with 64K L1 caches the average global region miss ratio varies from 79% down to 58%. Individual program behavior varies greatly as we will see in section 6.1.

These results suggest that if we could detect or predict *a priori* that a shared memory request would result in global region miss then we could avoid broadcasts for anywhere between 88% and 34% of all requests depending on the specific organization. There are potential energy, latency and bandwidth benefits:

1. We may be able to reduce the bandwidth requirements by avoiding the snoops that would result in a global region misses. Such snoops are unnecessary.

2. We may be able to reduce energy by avoiding the tag lookups in all remote caches similarly to previous work [9,20]. Furthermore, if the interconnection fabric permits it, we may be able to avoid communicating will all remote nodes reducing energy even further. For example, in systems where broadcasts are implemented over a switch-based interconnect this may be possible [8,16].

3. Finally, by avoiding some snoops we may also reduce the latency of the corresponding memory requests.

Of the aforementioned potential applications we limit our attention to bandwidth and tag lookup energy reduction.

Previous work has also shown that a significant fraction of requests that hit in some remote caches rarely hit in all or even many of them [9,20,23]. In section 5.3, we validate this observation for region sizes other than a block or a page and explain how our techniques can exploit this behavior to avoid many of these tag lookups.

4 Related Work

RegionScout is orthogonal to some, exploits similar phenomena as some others but could benefit most of the predictors for coherence [13,14,16,17,18,21,22]. While some predictors may be able to capture the same behavior (e.g., by predicting the destination set or by using an "all or nothing" policy [17]) RegionScout obviates the need to predict the destination set for many requests. Accuracy may improve further since the same predictor resources could be used to capture the behavior of fewer requests. Furthermore, RegionScout can free up interconnection bandwidth which could be used by more aggressive speculative coherence protocols.

Previous work for snoop energy reduction relies on similar phenomena as RegionScout. The *Page Sharing Table* (PST), proposed by Ekman *et. al.*, uses vectors that identify sharing at the page level [9]. It can be thought of as a partial, distributed page-level directory. The PST is tightly coupled with the TLB but its page size can be different. The sharing vectors are collected and broadcast on a separate bus so that nodes with no block in the page can avoid snoops thus reducing energy consumption.The Page Size Information extension augments the bus by sending page size information thus allowing different page sizes (pages of 1K and 4K were considered). In rare cases the PST has to be turned off (recovering from this state requires flushing the caches) since it has to track all pages that are locally cached (which may not be possible when space in the PST is exhausted). While PST utilizes precise page-level information, RegionScout targets the frequent case of regions that are not shared at all (which is often the common case also). For this reason and because it utilizes imprecise information, RegionScout requires a single additional global signal and only surgical changes in the existing infrastructure. RegionScout never has to be turned off for correct operation and disassociates the choice of region size from the rest of the design. As we explain in section 5.3, RegionScout is also able to avoid tag-lookups when some but not all nodes have blocks in a region.

In *Jetty* proposed by Moshovos *et. al.*, each node avoids many snoop-induced lookups that would otherwise miss [10]. Nodes maintain two structures that respectively represent a subset of blocks that are not cached (exclusive Jetty) and a superset of blocks that are cached (inclusive Jetty). With Jetty the requesting node does not know in advance that a request would miss in remote nodes and broadcasting all requests is necessary. In contrast, RegionScout allows the requesting node to determine in advance that a request would miss in all other nodes and can work for region sizes other than a block. Advance knowledge of global region misses allows new optimizations (such as reducing bandwidth and energy in the interconnect) that are not possible with Jetty. As we explain in section 5.3, a structure used by RegionScout can be used as simplified inclusive Jetty avoiding tag-lookups for requests that hit in some but not all remote caches. Because RegionScout can use regions that are much larger than a block much smaller structures are sufficient to capture most of the benefits.

Saldanha and Lipasti proposed serial snooping to reduce energy in shared multiprocessors where nodes are probed one after the other [23]. Ekman *et. al.,* evaluated Jetty and serial snooping for chip multiprocessors demonstrating little or no benefit [10].

5 RegionScout Filters

RegionScout filters allow each node to *locally* determine that a request would result in a global region miss and thus avoid the corresponding remote snoops and transactions. Informally, whenever a node issues a memory request it also asks all other nodes whether they hold any block in the same region. If they do not, then it records the region as not shared. Next time the node requests a block in the same region it knows that it does not need to probe any other node. Correctness is maintained since whenever another node requests a block in the same region, it will broadcast its request invalidating the not shared region records held by other nodes. As we will explain, what allows RegionScout to be effective yet inexpensive is that it works for *most* and not all requests that would result in a global region miss.

Formally, the RegionScout filters comprise two structures local to each node: (a) a *not shared region table*, or *NSRT*, and (b) a *cached region hash*, or *CRH*. The NSRT is a very small, cache-like structure that records regions that are known to be not shared. The CRH is a Bloom filter (similar to the inclusive Jetty [20]) that records a *superset* of all regions that are locally cached. Example organizations of CRH and NSRT are given in section 5.2.

Here is how RegionScout works: Initially, all caches, CRHs and NSRTs are empty. Whenever a node *N* issues a memory request, the other nodes respond normally via the existing coherence protocol but in addition they also report (for example via a separate wired-OR bus signal - see section 5.2.3 for a discussion) whether they cache any other block in the block's region. As we will explain shortly, nodes use the CRH to determine whether they hold a block in the specific region. Node N then records the region in its NSRT thus identifying it as not shared. Next time node N requests a block it first checks its NSRT and if a valid record is found then it knows that no other node holds this block and can avoid broadcasting this request. To ensure correctness it is imperative to invalidate NSRT entries whenever any other node requests a block in the corresponding region. This is easily done as part of the exiting protocol actions. Specifically, if another node *N'* requests a block in the same region, it too will check its own NSRT and since it will not find a valid record it will broadcast its request to all other nodes. The NSRT of node *N* will then invalidate the corresponding entry and subsequent requests will be broadcast as they should.

Key to RegionScout's operation is the ability at each node and given a block to determine whether there is any other block in the same region that is locally cached. One possibility would be to use a table to keep a *precise* record of all regions that are locally cached similarly to what is done for pages in PST [9]. The size of the table would artificially limit what can be cached and special actions would required to avoid exceeding these limits. RegionScout avoids these issues by using the CRH, an *imprecise* representation of the locally cached regions.

Without the loss of generality we limit our attention to the LocalL2 model. CRH works as follows: Whenever a block is allocated in or evicted from the L2, we use the region part of its address and hash into the CRH. Because there are much fewer CRH entries than cache blocks many regions may map onto the same CRH entry. Each CRH entry is conceptually a counter (we will see that a much simpler implementation is possible in section 5.2.2) that records the number of locally cached blocks in the regions that hash to it. Accordingly, when a block is allocated in the L2 we increment the corresponding CRH entry and when a block is evicted we decrement it. These updates are local at each node. Given a remote request for a block, the CRH can be used to indicate whether it would result in a region miss. Using the block's region we hash into the CRH and read the counter. If it is zero then we know for sure that no block of the specific region is locally cached. If it is non-zero then blocks in the same region *may be* cached (since many different regions map onto the same entry). It is the uncertainty of the latter response that allows us to use a small and effective structure for the CRH.

Figure 3 shows an example of RegionScout at work.

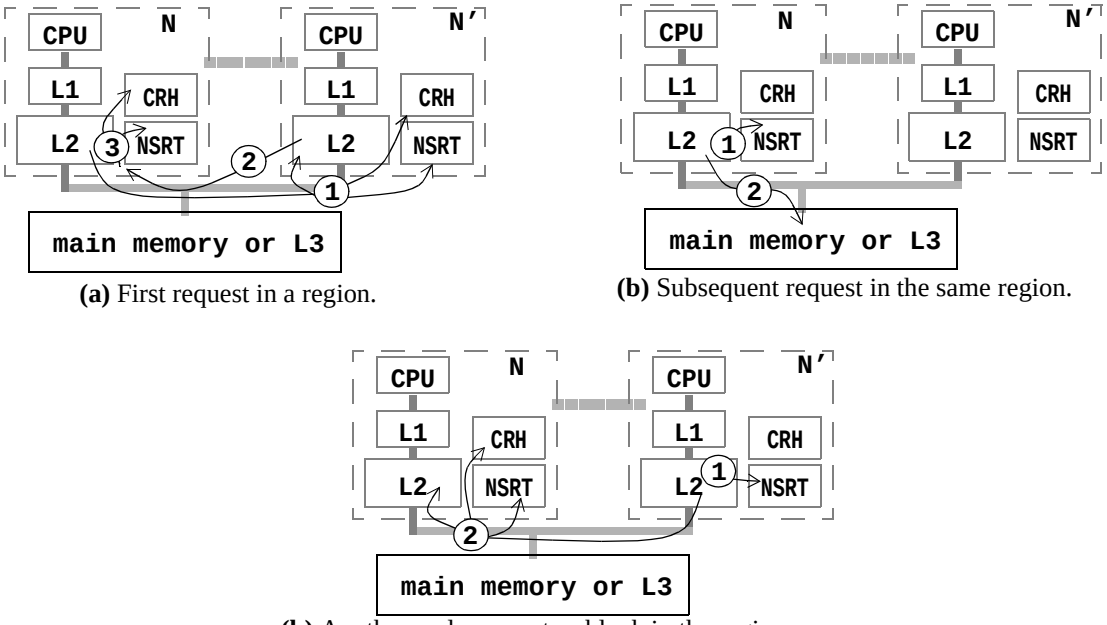
5.1 RegionScout as a Layered Extension

RegionScout can be implemented as a layered extension over existing coherence protocols and multiprocessor systems. No changes are required in the underlying coherence protocol, cache hierarchy and software. RegionScout can operate in parallel with existing structures, inhibiting snoops and broadcasts by intervening when necessary. To existing mechanisms it appears as if the coherence protocol has reported a miss. RegionScout is also completely transparent to software and the operating system. It does not impose any artificial limits on the choice of region size or on what can be locally cached. Finally, since it uses dynamically collected information it can adapt to changing sharing behavior identifying regions that are only temporarily not shared. Since every request has to probe the NSRT prior to being issued on the bus the latency of handling such requests will increase. The NSRT is comparatively tiny (we consider NSRTs of up to 64 entries) and as such its latency will be comparatively small. As we explain in section 5.3 the CRH can also be used as a simplified inclusive JETTY filter filtering many snoop-induced tag lookups that would otherwise miss.

5.2 Structures

5.2.1 NSRT and CRH

Figure 4 illustrates implementations of NSRT and CRH and how physical addresses are used to index them (we assume a 42 bit physical address space and 16K regions). NSRT is a simple table with entries comprising a valid bit (V) and a region tag (the upper part of the address). The NSRT can be set-associative. Prior to issuing a bus request each node checks whether a record exists in its NSRT. If so, it knows that this block is not shared. NSRT entries are evicted either as a result of limited space or when a remote node requests a block in a matching region. As we show in



(a) First request in a region.

(b) Subsequent request in the same region.

(b) Another node requests a block in the region.

Figure 3: An example illustrating how RegionScout works. Without the loss of generality we use the LocalL2 model and show only two nodes. (a.1) Node N requests block B in region RB. The request is broadcast to all nodes (N first checked its NSRT and since it was empty it found no matching entry for region RB). (a.2) All remote notes probe their CRH and report that they do not cache any block in region B. (a.3) Node N records RB as not shared and increments the corresponding CRH entry. (b.1) Node A is about to request block B' in region RB and first checks that its NSRT. (b.2) Since an entry is found the request is send only to main memory. (c.1) Node N' requests block B" in region RB. It first checks its NSRT. (c.2) Since the region is not recorded in its NSRT, it broadcasts its request. Node N invalidates its NSRT entry since now RB is shared.

the evaluation, a very small NSRT is sufficient. For example, in our experiments the largest NSRT we use has 64 entries. With the assumptions of this section, that NSRT requires less than 2K bits.

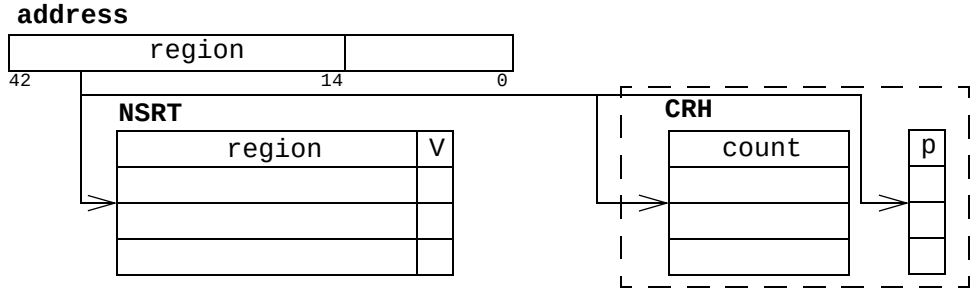


Figure 4: NSRT and CRH organization for 16Kbyte regions.

CRH is a table whose entries comprise a counter (*count*) field and a present bit (*p*). Essentially it is a inclusive Jetty with just one array [20] or a Bloom filter implementation [5]. The count field counts the number of matching blocks in the regions that map onto it. In the worst case, all blocks in the cache would belong to the same region. Hence, the count field needs *lg(Cache Size/Block Size)* bits. The p-bit indicates whether count is non-zero. We use the p-bits to reduce energy and delay when probing the CRH. As in the inclusive jetty the p-bit array is separate from the count array. Probing the CRH requires accessing the p-bit array only. Updating the CRH is done when blocks are allocated or evicted from the L2 or the L1 for models LocalL2 and SharedL2 respectively. Only when a count entry changes value from or to zero the p-bits are updated. Relatively small CRHs are sufficient for our purposes. For example, a 256 entry CRH needs 256 bits for the p-bit array and 1Kbits for the counter array (this is independent of region size).

5.2.2 Simplified CRH Counter Array

In the inclusive Jetty design the count fields are updated arithmetically. There are other ways to keep a count of matching blocks. We propose a simpler design that replaces the adder with a *linear feedback shift register* or *LFSR* [2]. Appropriately designed LFSRs of n-bits are capable of generating sequences of 2^n-1 states. LFSRs are used as pseudo-random number generators in many applications including testing and communications. LFSRs are much simpler, faster and energy efficient than arithmetic counters. For example, for a 4Mbyte L2 cache (worst case scenario we considered) we need 2^{16} states or 17 bit LFSRs. Such an LFSR requires just four XOR gates or few tens of transistors. Since the all zeroes combination does not appear in the maximum LFSR sequence a different encoding must be used for zero count (e.g., all ones).

5.2.3 Detecting Global Region Misses and Inhibiting Snoops

In a true bus implementation, an additional, wired-OR bus signal, *RegionHit,* can be used to identify global region misses. This represents a small overhead compared to typical bus implementations that use several tens of signals (e.g., approximately 90 in MIPS R10000 [11]). Prior to issuing a request RegionHit is de-asserted. In response, a node whose CRH reports a region "hit" asserts RegionHit. On global region misses no node will assert RegionHit. To inhibit snoops in a true bus implementation RegionHit can be overloaded as follows: Prior to issuing a request that would otherwise result in a global region miss (as reported by the NSRT) a node asserts RegionHit. Other nodes can sample RegionHit prior to snooping and thus avoid snooping altogether. For other requests RegionHit is de-asserted as before so that other nodes can snoop and report region hits. The information necessary for detecting global region misses and inhibiting snoops is a single bit. For this reason we expect that it will represent a small overhead for other interconnect architectures also. Moreover, it may be possible to embed this information in the existing protocol (i.e., in the control information) with no physical overhead.

5.3 Avoiding Tag Lookups for Non-Global Region Misses

As described RegionScout can avoid broadcasts for those requests that would result in a global region miss. Additional benefits are possible "for free" for requests that would result in region misses in *some* and not all remote nodes. The CRH can be used as a simplified inclusive Jetty as follows: Whenever node N makes a request all other nodes probe their CRHs to determine whether they will report a region miss. If a node N' determines that it has no matching block in the same region then it does not need to probe its local L2 or L1 tag array (for LocalL2 and SharedL2 respectively). By avoiding these lookups we can further reduce tag energy and tag bandwidth requirements. This comes at the expense of increased latency while probing that local tag arrays in response to a remote request. Since, probing the CRH amounts to probing the p-bit array, the latency penalty will be small.

How much potential is there for this optimation? In the interest of space we limit our attention to the 16K byte region and to four node LocalL2 and SharedL1 systems with 512K L2 and 64K L1 caches respectively. Table 2 reports the remote region hit count distribution. The fist column corresponds to global regions misses. The fraction of requests that incur a region miss in some remote caches (columns "1" and "2") is significant for all programs. In barnes, radiosity, radix, raytrace, volrend and to a lesser extend fmm many requests result in a region hit in all other nodes (column "3"). This may seem to contradict previous findings that most requests miss in remote caches but we emphasize that here we look at a region that is significantly larger than a block or a page.

Table 2: Remote region hit count distribution for four node LocalL2 and SharedL2 systems with 512K L2 and 64K L1 caches respectively. The region is 16Kbytes. Column "0" corresponds to global region misses.

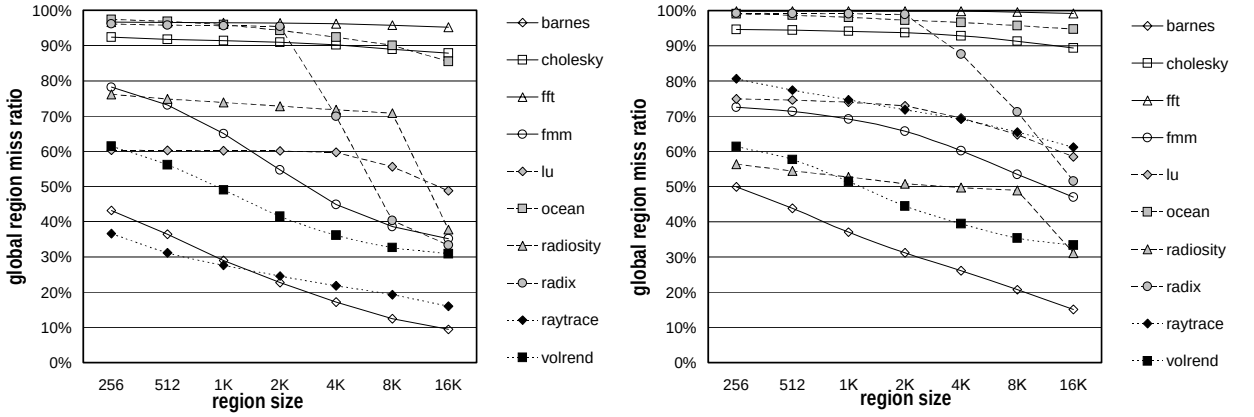
	Four Nodes, 16Kbyte Regions							
	LocalL2 Region Hit Count (%)				SharedL2 Region Hit Count (%)			
	0	1	2	3	0	1	2	3
barnes	9.38	13.89	12.84	63.89	15.12	14.03	10.61	60.24
cholesky	87.87	6.22	4.88	1.03	89.36	7.88	2.27	0.49
fft	95.22	4.47	0.19	0.12	99.14	0.74	0.02	0.09
fmm	35.20	28.22	19.69	16.9	47.00	21.32	5.61	26.06
lu	48.78	44.95	3.73	2.55	58.45	39.28	1.50	0.77
ocean	85.52	11.77	1.66	1.05	94.75	3.73	1.26	0.26
radiosity	37.75	35.56	4.15	22.54	31.10	19.07	1.97	47.86
radix	33.32	15.15	25.99	25.54	51.55	28.13	15.24	5.09
raytrace	16.01	10.86	28.23	44.90	61.16	23.24	9.80	5.80
volrend	30.94	5.07	37.87	26.13	33.31	9.72	26.96	30.01
average	47.99	17.61	13.92	20.46	58.09	16.71	7.52	17.66

6 Evaluation

In the interest of space and unless otherwise noted we limit our attention to the LocalL2 and SharedL2 models with 512K L2 and 64K L1 caches respectively. In section 6.1 we show the global region miss behavior of individual programs as a function of region size. In section 6.2 we demonstrate that practical RegionScout filters can capture many global region misses. We chose a large enough NSRT and focus on region and CRH size. We identify the trade offs among region size, detected global region misses and RegionScout filter storage requirements. In section 6.3 we demonstrate that RegionScout can be used to avoid broadcasts in snoop coherence while in section 6.4 we show that it can be used to avoid many snoop induced tag lookups thus reducing tag energy. The two applications also serve to demonstrate that we can tailor the RegionScout filters to meet different trade-offs.

6.1 Per Program Global Region Miss Behavior

On the average there are many global region misses as we have seen in section 3. Since average behavior can be misleading we also look at individual program behavior. Figure 5 reports the global region miss ratio per program for regions of 256 through 16K bytes. Parts (a) and (b) are for models LocalL2 and SharedL2 respectively. In the interest of space we restrict our attention to four node systems. In addition to the intrinsic sharing characteristics of each program the observed global miss ratios are mostly reversely proportional to the region and cache sizes. Programs can be classified in three categories based on how sensitive they are to the choice of region size. Cholesky, fft and ocean exhibit high global region miss ratios and are mostly insensitive to region (and cache) size. In barnes, fmm, raytrace and volrend the global region miss ratio decreases almost linearly as the region size increases. Finally, radix, radiosity and to a lesser extend, lu, exhibit abrupt changes in their global miss ratio when the region size increases above 2K, 8K and 4K respectively. For the smallest region size of 256 bytes the global miss ratio is above 37% for all programs under both models. With the largest region of 16K bytes the miss ratio remains above 30% for all programs except barnes and raytrace under the LocalL2 model (and under the SharedL2 for barnes). This result is important as it suggests that sufficient global misses occur even when we look at very coarse grain sharing. Using large regions is attractive as smaller structures could be used to track them.



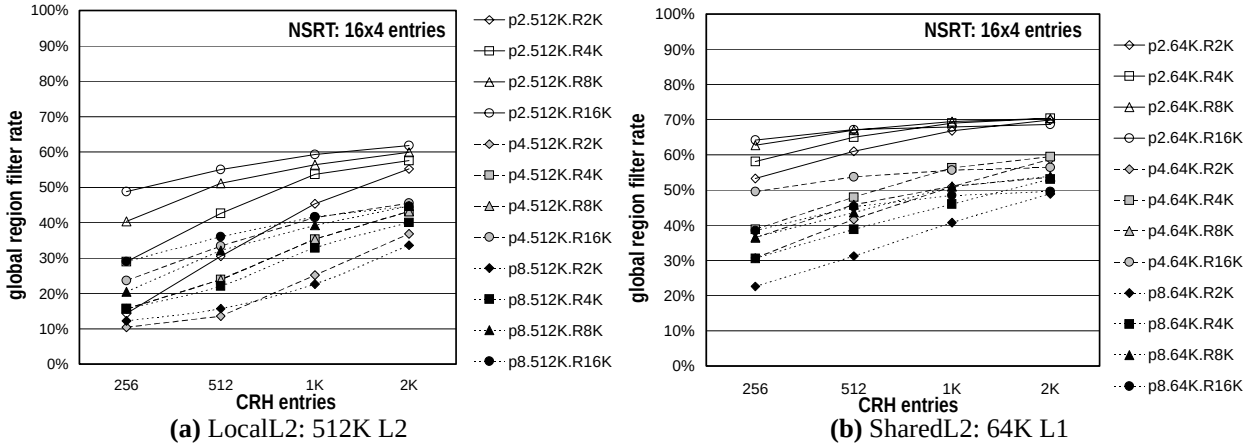
(a) LocalL2: 4 nodes, 512K L2 **(b)** SharedL2: 4 nodes, 64K L1

Figure 5: Global Region Miss ratios per program for various region sizes.

6.2 Can Practical RegionScout Filters Detect Many Global Region Misses?

We have seen that sufficient global region misses occur in the programs we studied. The question we answer affirmatively in this section is whether practical RegionScout filters can capture most of them. For this purpose we use an NSRT with 64 entries (16 sets, 4-way set associative) and measure the filter rate for different CRHs. We define the *filter rate* as the fraction of global region misses that are detected by the RegionScout filter. A global region miss is detected when the originating node finds a matching entry in its NSRT prior to issuing the request. We found that an NSRT of 64 entries approximates one with infinite entries. We consider CRHs of 256 through 2K entries and regions of 2K through 16K bytes. The resulting average filter rates are shown in figure 6. Each of the curves corresponds to a different region size and to a system with a different number of nodes. The curves are identified as pN.CRS where N is the number of nodes, C is the cache size (L2 for LocalL2 and L1 for SharedL2) and S is the region size. While we

have seen that using smaller regions typically results in higher global miss ratios here we observe a trade-off between region and CRH size. In most cases using a larger region results in a higher filter rate with the difference being greater for the smaller CRHs (i.e., 256 or 512 entries). The CRH is an imprecise representation of cached regions. Using larger regions results in fewer regions and improves the CRH's ability to separate among them. Only when the size ratio "CRH over cache" becomes large enough we see a benefit in using smaller region sizes. This can be seen under the SharedL2 model when the local caches are only 64K and for the 2K entry CRH.



(a) LocalL2: 512K L2 **(b)** SharedL2: 64K L1

Figure 6: Average global region miss filter rates (see text for the definition) for various CRH sizes (X-axis). The NSRT has 64 entries organized in 16 sets of four entries each. The curves correspond to systems with different number of nodes *N* and to different region sizes *S* as identified by the p*N*.*C*R*S* labels. *C* is the size of the L2 and L1 caches for models LocalL2 (a) and SharedL2 (b) respectively.

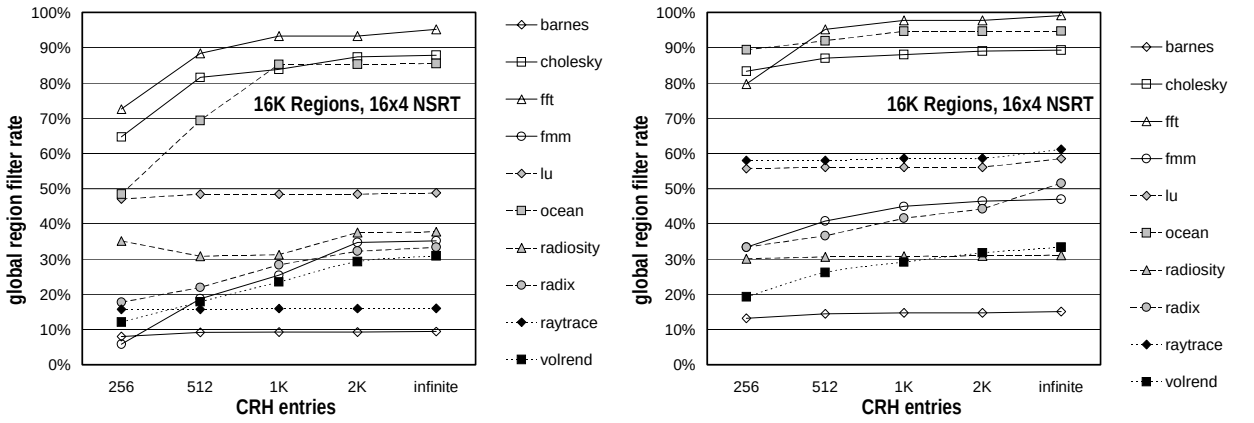
Figure 7 shows the filter rates per program for various CRH sizes. We restrict our attention to the 16K byte regions and to four node systems with 512K L2 and 64K L1 caches. Under the SharedL2 model (part (b)) it is conflict misses that dominate traffic and this results in much higher temporal and spatial locality in the request stream. For this reason and given that there are fewer regions per node even the 256 entry CRH can capture most global region misses. While the filter rates increase with CRH size these improvements are modest. Under the LocalL2 model in part (a), the filter rate is more sensitive to CRH size. This is because a much larger portion of data and thus more regions remain resident in the caches. Under LocalL2, larger CRHs result in significantly higher filter rates for some programs (e.g., ocean, cholesky and fmm). Lu, barnes and raytrace are mostly insensitive to CRH size. An anomaly is observed for radiosity where the filter rate decreases when the CRH entries are increased from 256 to 512. In this case we have thrashing in the NSRT: more regions are identified as non-shared but their base addresses are such that they trash few sets in the NSRT. This thrashing persists for the 2K CRH but the resulting filter rate is higher since additional regions are identified as non-shared which map to different sets in the NSRT.

Summarizing our findings we have found that practical RegionScout filters can detect most global region misses. While there are more global region misses for smaller regions, detecting them requires larger RegionScout filters. For the configurations we studied, using a large region (e.g., 16K) is typically better. For a given region size, we can improve the filter rate if we use a larger CRH. In the next two sections we present applications of RegionScout filters where we exploit the trade off among region size, filter rate and RegionScout storage requirements to better suit each application.

6.3 Bandwidth Reduction

We demonstrate that RegionScout can be used to avoid broadcasts in snoopy coherence systems. We do not show that this leads to some other tangible benefits. Potential benefits include a reduction in average memory latency (and performance) and in energy in the interconnect. The freed bandwidth could also be used in more fruitful ways (e.g., for speculation of various forms such as coherence speculation and prefetching).

Emerging technology trade-offs favor point-to-point links over true busses for high performance interconnects [8,16]. Accordingly, we model a point-to-point interconnect where nodes are connected via a switch. Such an interconnect is practical for small scale multiprocessors. Under these assumptions a broadcast requires *N-1* messages in an *N*



(a) LocalL2: 4 nodes, 512K L2 **(b)** SharedL2: 4 nodes, 64K L1

Figure 7: Filter rates per program for 16K byte regions on four node systems with 512K L2 (a) and 64K L1 (b) caches respectively. The NSRT is fixed at 64 entries (16 sets, 4-way set-associative) and the number of CRH entries varies as shown on the X-axis.

node system (more would have been required for other interconnects such a ring or a torus). Each link is capable of transferring 64-bits of data per message plus address and control information. Transferring a 32 byte block requires five messages at minimum (one is for the request and the other four are for the data). In a system with a RegionScout filter it is not necessary to probe all other nodes for those requests that the NSRT identifies as global region misses. As a metric of bandwidth we count the number of messages sent during execution with and without RegionScout and report the ratio:

$$MessageRatio = \frac{MessageCountWithRegionScout}{MessageCountWithoutRegionScout}$$

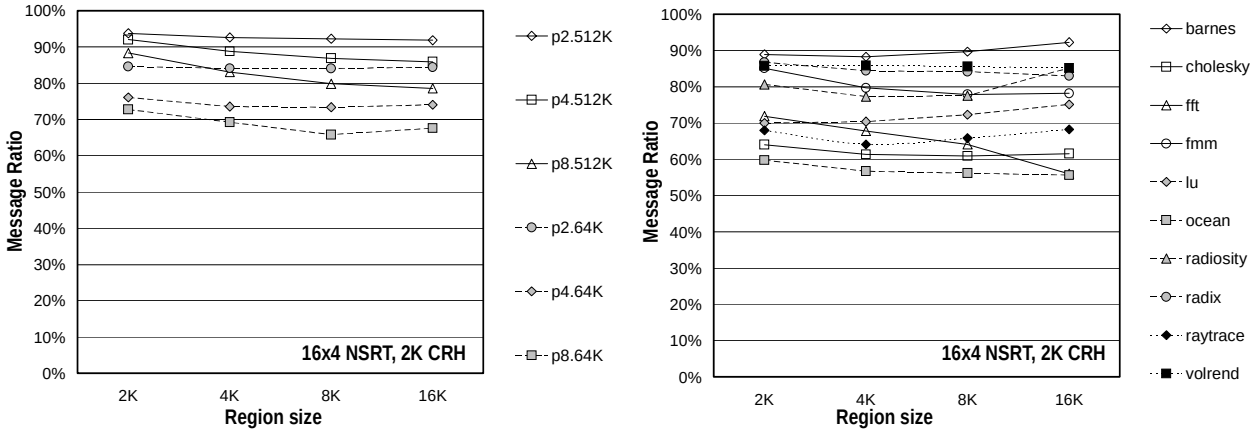
For this application we can afford to use the larger RegionScout filters. Accordingly, the NSRT has 64 entries and is four-way set-associative and the CRH has 2K entries. Figure 8(a) reports the message ratio for regions of 2K up to 16K and for LocalL2 and SharedL2 systems with different number of nodes. Systems are identified as pN.C where N is the number of nodes and C is the cache size (512K L2 and 64K L1 for LocalL2 and SharedL2 respectively). We observe a reduction in messages in the range of 6% up to 34%. The reduction is higher for the SharedL2 system (grey marks) for two reasons: Coherence messages are a higher percentage of all messages due to the smaller cache block size and the filter rates are higher as compared to the LocalL2 systems. In some cases, using larger regions results in higher benefits since it allows the CRH to better separate among the fewer regions. For the configurations we studied using smaller regions does not result in significantly higher benefits even though we have seen that there are more global misses with smaller regions. This suggests that the CRH configurations we studied are incapable of separating among many small regions and better suited for larger regions.

For completeness we report per program message ratios for the four node SharedL2 system in figure 8(b). Individual program behavior varies as the resulting behavior is a function of the filter rate and the fraction of messages used for coherence. Programs with a higher fraction of coherence messages are more sensitive to changes in the filter rate.

6.4 Tag Energy Reduction

RegionScout filters can be used to avoid remote tag lookups (snoops) for those requests that will result in a global region miss and are identified by the requestor's NSRT. The CRH can also be used as a simplified JETTY avoiding some of the tag lookups that will result in a region miss in some remote caches. Previous work has shown that snoop induced tag lookups represent a significant fraction of cache energy [9,10,20]. We restrict our attention to 16Kbyte regions and the LocalL2 and SharedL2 systems with 512K L2 and 64K L1 caches. Since the RegionScout filters consume energy they represent an overhead which we would like to minimize. Accordingly, we select small RegionScout filters. The NSRT has 16 entries and is direct mapped and the CRH has 256 entries. We measure the energy consumed during tag lookups in the L2 and L1 caches (all accesses, local or snoop induced) and report the ratio:

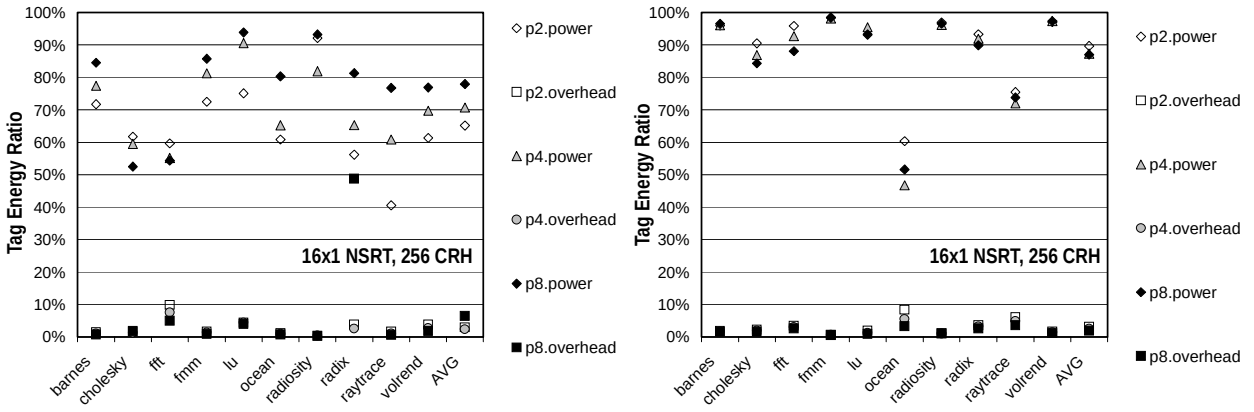
$$TagEnergyRatio = \frac{TagEnergyWithRegionScout + RegionScoutEnergyOverhead}{TagEnergyWithoutRegionScout}$$



(a) Average Message Ratio **(b)** SharedL2: 4 nodes, 64K L1. Message Ratio per Program

Figure 8: Avoiding broadcasts with RegionScout for various region sizes. NSRT is 64-entries, 4-way set-associative and the CRH has 2K entries. Shown is the ratio of messages sent with RegionScout over those sent without it. (a) Average message ratio for LocalL2 and SharedL2 systems with 512K L2 and 64K L1 caches respectively. (b) Message ratio per program for the SharedL2 system with 64K L1 caches.

To measure energy we use the WATTCH models [7]. The L1 and L2 caches and the RegionScout structures are split into four banks. Figure 9 reports the tag energy ratio per program and for systems with different number of nodes. The RegionScout filters energy overhead is included in the ratio and also reported separately. The resulting energy savings are primarily a function of (1) the fraction of tag accesses that are snoop-induced versus those that are generated locally, (2) the global region miss filter rate and (3) the filter rate of each CRH for those requests that are not identified as global region misses. Energy savings are modest. In some cases the energy savings are higher than the filter rate for global region misses. This suggests that the CRHs filter many snoops that are not identified as global region misses by RegionScout. Most of these tag lookups are for requests that result in region misses in some but not all nodes (as per the discussion of section 5.3) and the others are for global region misses that are not identified by the requestor's NSRT. The LocalL2 systems benefit more from RegionScout since there snoop-induced tag lookups represent a higher fraction of overall tag accesses. This result corroborates previous findings [9,10]. RegionScout overhead is typically low with radix being a noticeable exception. Since we did not have an appropriate interconnect energy model these results do not include the energy savings on the interconnect.



(a) LocalL2: 512K L2, 16K regions. **(b)** SharedL2: 64K L1, 16K regions.

Figure 9: Tag energy ratio and RegionScout energy overhead. NSRTs have 16 entries and are direct-mapped. The CRH has 256 entries. Region is 16K bytes.

7 Summary

We were motivated by the observation that many requests in shared memory multiprocessors not only do not hit in any remote node but also do not find any other block in a much larger surrounding region (global region miss). We proposed RegionScout a family of small and effective filters that can detect most of the requests that would result in a global region miss. RegionScout filters utilize imprecise information about the regions that are cached at each node in the form of hashed bit vector. This has several advantages as it requires only surgical additions over the existing infrastructure of coventional shared multiprocessors. RegionScout filters are transparent to software and do not impose any artifical limits on what can be cached and when. We have demonstrated that RegionScout filters can be used to reduce bandwidth by avoiding broadcasts for some requests. Moreover, we have shown that can reduce energy by avoiding some of the tag lookups for snoops that would miss.

8 References

[1] ___, MIPS R10000 Microprocessor User's Manual v2.0, MIPS Technologies, Inc., January 1997.

[2] M. Abramovici, M. A. Breuer, A. D. Friedman, Digital Systems Testing & Testable Design, Wiley-IEEE Computer Society Press, January 1993.

[3] P. Bannon, B. Lilly, D. Asher, M. Steinman, D. Webb, R. Tan, and T. Litt. Alpha 21364: A Single-Chip Shared Memory Multiprocessor, Government Microcircuits Applications Conference 2001, Digest of Papers, Defense Technical Information Center, Belvoir, Va., March 2001.

[4] L. A. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, and B. Verghese. Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing. In *Proceedings of the 27th Annual International Symposium on Computer Architecture,* June 2000.

[5] B. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of ACM*, pages 13(7):422-426, July 1970.

[6] D. Burger and T. Austin. The Simplescalar Tool Set v2.0, *Technical Report UW-CS-97-1342. Computer Sciences Department, University of Wisconsin-Madison*, June 1997.

[7] D. Brooks, V. Tiwari M. Martonosi. Wattch: A Framework for Architectural-Level Power Analysis and Optimization. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, June 2000.

[8] W. J. Dally and J. W. Poulton. Digital Systems Engineering. Cambridge University Press, 1998.

[9] M. Ekman, F. Dahlgren, and P. Stenström, TLB and Snoop Energy-Reduction using Virtual Caches for Low-Power Chip-Multiprocessors. In *Proceedings of ACM International Symposium on Low Power Electronics and Design*, August 2002.

[10] M. Ekman, F. Dahlgren, and P. Stenström: Evaluation of Snoop-Energy Reduction Techniques for Chip-Multiprocessors. In *Proceedings of the First Workshop on Duplicating, Deconstructing, and Debunking (WDDD-1)*, May 2002.

[11] L. Hammond, B. Hubbert, M. Siu, M. Prabhu, M. Chen, and K. Olukotun. The Stanford Hydra CMP, *IEEE MICRO Magazine*, March-April 2000.

[12] J. Huh, D. Burger, and S. W. Keckler. Exploring the design space of future CMPs. *In Proceedings 10th International Conference on Parallel Architectures and Compilation Techniques*, September 2001.

[13] S. Kaxiras and C. Young. Coherence Communication Prediction in Shared-Memory Multiprocessors. In *Proceedings of the Sixth IEEE Symposium on High-Performance Computer Architecture*, Jan. 2000.

[14] A.-C. Lai and B. Falsafi. Memory Sharing Predictor: The Key to a Speculative Coherent DSM. In *Proceedings of the 26th Annual International Symposium on Computer Architecture*, May 1999.

[15] N. Manjikian, Multiprocessor Enhancements of the SimpleScalar Tool Set, *ACM Computer Architecture News, Vol. 29, No. 1*, March 2001.

[16] M. M. K. Martin, M. D. Hill, and D. A. Wood. Token Coherence: Decoupling Performance and Correctness. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, June 2003.

[17] M. M. K. Martin, P. J. Harper, D. J. Sorin, M. D. Hill, and D. A. Wood. Using Destination-Set Prediction to Improve the Latency/Bandwidth Tradeoff in Shared-Memory Multiprocessors. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, June 2003.

[18] M. M. K. Martin, D. J. Sorin, M. D. Hill, D. A. Wood, Bandwidth Adaptive Snooping, In *Proceedings of the 8th International Symposium on High- Performance Computer Architecture*, January 2002.

[19] J. M. Mellor-Crummey and M. L. Scott. Algorithms for Scalable Synchronization on Shared-Memory Multiprocessors, *ACM Transactions on Computer Systems*, February 1991.

[20] A. Moshovos, G. Memik, B. Falsafi, and A. Choudhary. JETTY: Filtering snoops for reduced energy consumption in SMP servers. In *Proceedings of the 7th International Symposium on High- Performance Computer Architecture,* January 2001.

[21] S. S. Mukherjee and M. D. Hill. Using Prediction to Accelerate Coherence Protocols. In *Proceedings of the 25th Annual International Symposium on Computer Architecture,* June 1998.

[22] J. Nilsson, A. Landin, P. Stenström. Coherence Predictor Cache: A Resource Efficient Coherence Message Prediction Infrastructure. In P*roceedings of the 6th IEEE International Symposium on Parallel and Distributed Processing Symposium*, April 2003.

[23] C. Saldanha and M. H. Lipasti, Power Efficient Cache Coherence, *High Performance Memory Systems*, edited by H. Hadimiouglu, D. Kaeli, J. Kuskin, A. Nanda, and J. Torrellas, Springer-Verlag, 2003.

[24] J. M. Tendler, J. S. Dodson, J. S. Fields, Jr., H. Le, and B. Sinharoy. POWER4 system microarchitecture. *IBM Journal of Research and Development, vol. 46, No 1*, January 2002.

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

[26] D. A. Wood and M. D. Hill. Cost-effective parallel computing. IEEE Computer Magazine, 28(2), Feb. 1995.