Using Partial Tag Comparison in LowPower Snoopbased Chip Multiprocessors

Ali Shafiee, Narges Shahidi, Amirali Baniasadi

Using Partial Tag Comparison in Low-Power Snoop-based Chip Multiprocessors

Ali Shafiee[♠] Narges Shahidi[♠] Amirali Baniasadi[†]
[♠]CE Department, Sharif University of Technology, Tehran, Iran
[†]ECE Department, University of Victoria, Victoria BC, Canada
{a_shafiei,shahidi}@ce.sharif.edu amirali@ece.uvic.ca

Abstract—In this work we introduce power optimizations relying on partial tag comparison (PTC) in snoop-based chip multiprocessors. Our optimizations rely on the observation that detecting tag mismatches in a snoop-based chip multiprocessor does not require aggressively processing the entire tag. In fact, a high percentage of cache mismatches could be detected by utilizing a small subset but highly informative portion of the tag bits.
Based on this, we introduce a source-based snoop filtering mechanism referred to as S-PTC. In S-PTC possible remote tag mismatches are detected prior to sending the request. We reduce power as S-PTC prevents sending unnecessary snoops and avoids unessential tag lookups at the end-points. Furthermore, S-PTC improves performance as a result of early cache miss detection.
S-PTC improves average performance from 2.9 % to 3.5% for different configurations and for the SPLASH-2 benchmarks used in this study. Our solutions reduce snoop request bandwidth from 78.5% to 81.9% and average tag array dynamic power by about 52%.

1. Introduction

The continuous downscaling of transistor dimensions together with limitation on program's instruction level parallelism has popularized shared-memory chip multi-processor (CMP) architecture as an effective solution. On a CMP, cores communicate through shared variables and based on cache coherence protocols. Cache coherence protocols facilitate propagating the recently updated values to all concerning caches [1]. In addition, cache coherency provides cores with the latest value of the requested shared variables. The delay associated with the coherency mechanisms postpones shared variable updates and read operations. Accordingly, one way to enhance the overall CMP performance is to speedup the coherence process.
To reduce coherency delay, commercial small-scale [2] and possibly larger CMPs [3] exploit Snoopy Cache Coherence (SCC) protocols. SCC protocols take an aggressive approach and broadcast memory requests to all cores in the system. Unfortunately, SCC protocols impose high interconnect bandwidth demand and frequent unnecessary remote cache searches [3].
Previous research has introduced different approaches to solve the above problems. One possible approach exploits snoop filters to eliminate useless interconnect and memory activities [4-7]. Snoop filters come in two classes: source-based and destination-based. In source-based filters [4,5] each node decides *locally,* but based on *global* knowledge, to broadcast a message or not. While this approach can eliminate some unessential traffic, it cannot stop delivering messages or prevent cache lookups in non-concerning processors when it attempts to broadcast. Destination-based filters, however, focus on eliminating unnecessary lookups at destinations [6,7]. On the contrary to source-based filters, these filters rely on *local* knowledge to determine whether lookup is necessary or not. They take advantage of the snoop request access pattern locality [6] or bloom filters [7] to eliminate non-required lookups.
We extend previous work by using partial tag comparison (or simply **PTC**) in snoop-based chip multiprocessors. We rely on the observation that a considerable share of tag mismatches could be avoided by comparing a subset of tag bits, making an entire tag comparison unnecessary. We take advantage of this phenomenon and store a small number of tag bits for tags recorded in all cores in the source node to facilitate early mismatch detection. Prior to sending a snoop request, we compare the subset of address tag bits to those stored in the source node and avoid sending the snoop request to nodes showing a mismatch.
It should be noted that there are two classes of coherency cache misses: global and local. A global miss occurs when the requested address is missed in every remote cache. In the case of a local miss, while one or more cores miss the requested data, there is at least one remote cache that has a copy of the requested block. Previous suggested source-based filters focus on global misses. As we show in this paper, our proposed source-based PTC-based mechanism detects both global and local misses increasing the power reduction opportunities.
Using a small number of tag bits makes early cache miss detection possible. This results in performance improvement for some of the applications studied here. Therefore, while previously suggested techniques often save power at the expense of performance, we improve performance and power simultaneously for some applications.
In summary we make the following contributions.

- We show that it is possible to maintain cache coherency by using only a small number of tag bits. Our study shows that it is possible to detect, on average, between 95% to 98% of global and local

remote misses by taking into account only the eight lower bits of the requested tag in different CMP configurations.

- We propose source-based PTC (or S-PTC). S-PTC relies on storing a snapshot of the storage components involved in snooping at the source-side. S-PTC reduces interconnect bandwidth requirement (78.5% to 81.9%) and tag array dynamic power (52%) while improving average performance up to 3.5%.

The reset of the paper is organized as follows. In section 2 we discuss background. In section 3 we present our motivating findings. In section 4 we discuss S-PTC in more details. In section 5 we present methodology and results. In section 6 we discuss related work. Finally, in section 7 we offer concluding remarks.

2. Background

SCC protocols implementations consist of coherence communication followed by possible cache-to-cache data transfers. To keep coherence transactions fast, designers utilize different interconnects for different cache coherency operations [8]. Small-scale CMPs (which are the focus of this work) use *address*, *snoop* and *response/command* buses for sending requests to the *snoop controller*, broadcasting the requested addresses to all cores and gathering each core's response respectively [1]. In addition, the snoop controller could be integrated with the memory controller in order to manage requests going to or coming from the higher memory level. Small-scale CMPs could also assume a large separate pipelined bus for data communication [8]. Figure 1 illustrates a typical sequence of coherence transactions for coherence buses.

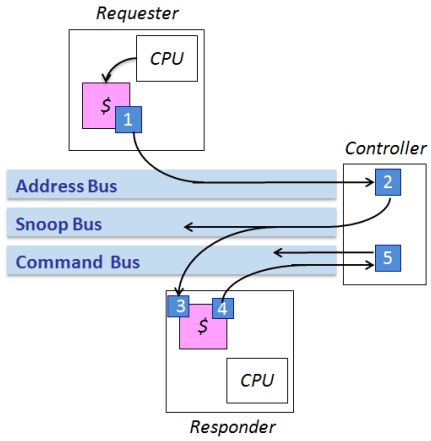

Figure 1: *The sequence of transactions for SCC protocols over shared bus fabrics. 1) One of the cores sends the requested address using the address bus. 2) The controller decodes the requested address and queues it at the end of the snoop queue. After it has reached the head of the queue, the controller broadcasts the requested address over the snoop bus. 3) Other cores search their cache tag array and prepare their response. 4) Every core puts the lookup result on the response bus. The controller receives the result. 5) The snoop controller makes and propagates the final decision according to the responses received.*

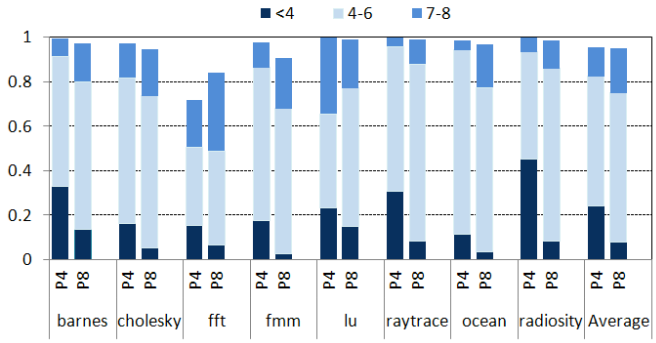

Figure 2: *How often using the n lower tag bits (n<4, 4≤n≤6 and 7≤n≤8) is enough to detect a remote cache miss in the configurations investigated in this work (see section 5 for details).*

3. Motivation

Conventional snooping relies on broadcasting and processing all tag bits. This is inefficient from the power and bandwidth point of view as it is possible to detect mismatches using a small portion of tag bits. To provide better insight, in figure 2 we report how often broadcasting the lower *n* bits of the tag is enough to detect a mismatch in all ways used in remote set associative caches. As reported, a large number of tag mismatches could be detected using a small number of bits.

On average, and for all configurations, more than 84% of snoop misses could be detected broadcasting only eight tag bits. Our study shows a similar observation for cache lookups (not reported here in the interest of space).

4. S-PTC: Cache Optimizations

In this section, we introduce S-PTC in more detail.

4.1 S-PTC

S-PTC detects useless coherence traffic and lookups prior to sending the snoop request. This is done by keeping a small subset of tag bits of all core storage states. This small subset is used to avoid sending snoop messages to cores that do not have a valid version of the requested line. To identify such cores, S-PTC holds a repository referred to as the S-filter containing the small subset of bits for every tag. This filter is accessed at the core initiating the snoop. The S-filter contains the least significant bits of every tag (LSB) for partial tag comparison.

S-PTC avoids snooping and searching all caches as it can identify those likely to have the requested block. By limiting the search to some and not all cores, S-PTC avoids processing both global and local misses.

4.2 S-PTC Updating

S-PTC requires keeping track of the storage states of snooping components. Therefore having an efficient filter updating mechanism is essential. To provide the S-filter with recent storage states, we need to keep all cores informed about every eviction and insertion event. Since evictions occur as a result of inserting new cache blocks, information regarding both events could be broadcasted simultaneously.

For example, assume that one of the local caches inserts block A, replacing block B. At this point the controller,

while inserting A into the cache, informs all filters regarding the latest position and state of B. It is important to note that the controller should also broadcast B's dirty bit.

An S-filter not informed about the dirty blocks could potentially mistake an outdated block residing in the upper level memory for a dirty block sitting in the writeback buffer. This could happen if an evicted block is requested before being written to the upper memory level. An uninformed S-filter assumes that all evicted blocks are clean. Consequently the coherency mechanism will look for the missing data in the upper level memory. To prevent such scenarios storing a low resolution snapshot of the writeback buffer (WB) is also needed.

To keep S-filters up-to-date the following hardware modifications are needed.

Local tag array: Local tag array should send the insertion position, (i.e., the way number or W), in set associative caches and the dirty bit (D) of the corresponding evicted line to all S-filters. In the case of direct mapped caches sending W is irrelevant.

Snoop controller: The snoop controller is responsible for sending the required information to all S-filters. Since the snoop controller has to accommodate the three new fields used by the address bus, we add them to the pending memory request table of the memory controller.

Besides the detecting bits, S-PTC stores a bit indicating the validity of tag and a parity bit. S-PTC does not use or modify the coherence bits of the cache blocks; therefore it is orthogonal to the underlying cache coherence protocol.

In figure 3 we present the steps taken during S-PTC updating in more detail.

Note that S-PTC relies on broadcasting the final command to all cores in order to update all S-filters.

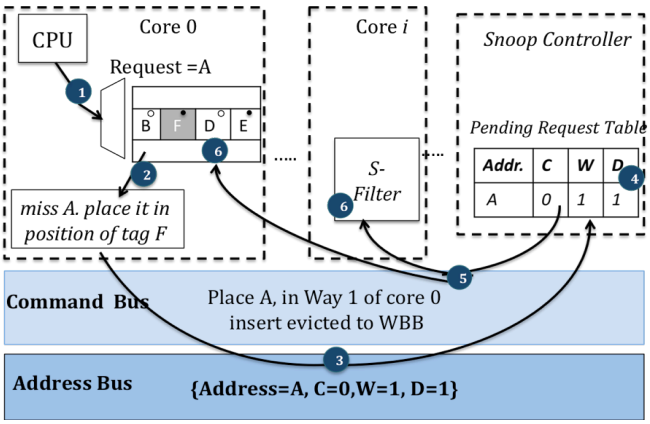

Figure 3: *S-PTC updating. 1) CPU requests address A in core 0. 2) The local tag array in core 0 signals a miss and selects the position of the dirty line (F) for replacement. 3) Upon granting access, core 0 sends a request to the snoop controller for address A containing position and the dirty bits of line F over the address bus. 4) Snoop controller stores information about the evicted line in the pending request table. 5) After locating line A, the controller broadcasts a message over the command bus a) to order core 0 to receive line A and b) to tell all S-filters about evicting F and inserting A in core 0. 6) Finally, core 0 replaces F with A, while forwarding F to WB. In addition, the S-filters are updated while they snoop the command bus to get the latest event.*

In the baseline snooping, however, the final command only addresses the requesting or providing cores. Further optimize of our solution is part of or our ongoing research.

5. Methodology and Results

In this section we report methodology and results. Section 5.1 explains methodology. In section 5.2 we report performance improvements. In section 5.3, 5.4 and 5.5 we report bandwidth reduction, tag power and area overhead respectively.

5.1 Methodology

We use a representative subset of SPLASH-2 [9] (see Table 1) applications. We use and modify the SESC simulator [10] to simulate our system. Table 2 reports the configurations used in our experiments. These configurations are consistent with today's implementation of CMPs [11]. We use CACTI 6.0 [12] to estimate cache tag array and filter power. We assume 65nm technology with a target frequency of 5GHz. Note that S-PTC uses auxiliary structures and therefore comes with timing and power overhead. In this study we take this overhead into account. We compare S-PTC to a conventional CMP.

5.2 Performance

In figure 4 we report relative performance for S-PTC compared to a conventional snooping system for the two configurations presented in Tables 2 and for the applications studied here. As reported, we witness an average performance improvement of 2.9% and 3.5% for 4-way and 8-way CMPs respectively.

Table 1: *SPLASH input*

benchmark	input	Benchmark	input	Benchmark	input
barnes	16k particle	radiosity	-batch -room	water-NSq	512 molecules
cholesky	K29.O	Radix	8M keys	water-sp	512 molecules
lu	512×512 matrix	raytrace	Balls4.env	fmm	16k particles
Ocean	258×258 grid	volrend	256×256×126 voxels	fft	-m20

Table 2: *CMP configurations*

Processor		Memory	Interconnect
Branch predictor : 16K Bimodal & gshare Penalty: 17 cycles	(instr. FX/FP windows size) 80/32 ROB size: 176	DL1/IL1 size: 64/32 KB DL1/IL1 Asso. : 4/2 ways Latency: 3 cycles Replace policy: WB/WT	4/8 way CMP fast Bus delay: 2/3 cycles Slow wire penalty: 2 cycles Bus arbitration: 6 cycles
Fetch/issue/commit: 4/4/5 RAS: 32 entries BTB: 2-way 2k entries	Int/FP register file :176 Max load/store: 62/56	Cache line: 64 B Protocol : MESI Memory Latency: 500 cycles	Bus size: Adr./Snoop/Cmd 7/12/8 Bytes [8]

Configuration	Description	L2	L2 Latency	Data Interconnect
P4	four cores private L1 shared L2	4 MB 4-banked 16-way	10 cycles	crossbar
P8	eight cores private L1 shared L2	8-banked 16-way with 512 KB per bank	10 cycles	crossbar

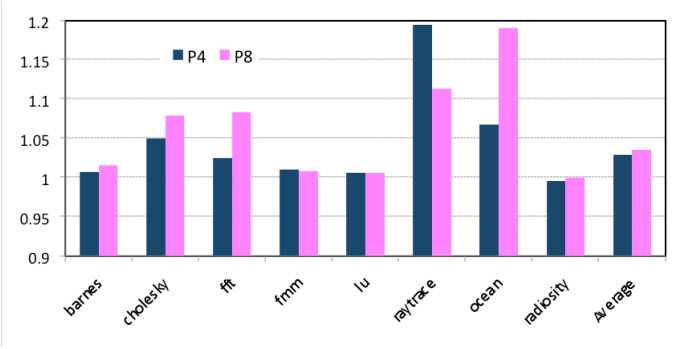

Figure 4: *Performance*

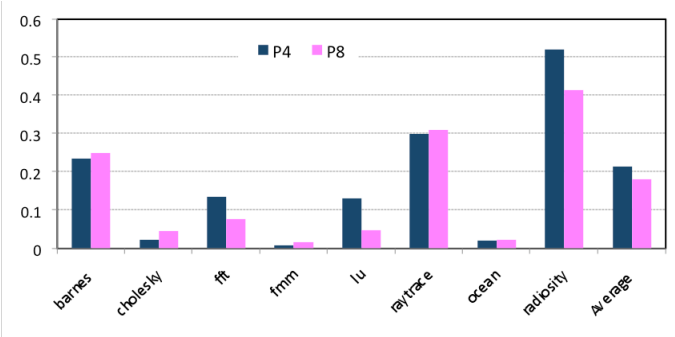

Figure 5: *Bandwidth utilization*

Note that raytrace and ocean show better performance improvements compared to other benchmarks. This is consistent with previously suggested studies [13]. We improve performance as remote cache misses are detected before reaching the end-points (i.e., at the requesting node), skipping several snoop stages and initiating an L2 access early.

5.3 Bandwidth Utilization Reduction

Source-based filtering reduces bandwidth requirements. The exact bandwidth reduction depends on the bus physical implementation details. For a simple single segment bus bandwidth reduction is equal to number of snoop requests filtered. For segmented or star buses S-PTC can reduce bandwidth further as it can eliminate some remote tag arrays searches when it has to snoop. Figure 5 reports the normalized bandwidth utilization for our two configurations. We consider two- and four- branch buses for 4-way and 8-way CMPs respectively. As reported, bandwidth reduction is 78.5% and 81.9% for 4-way and 8-way CMPs respectively.

5.4 Tag Lookup Power

In this section we report tag lookup power. S-PTC reduces dynamic power as it prevents unessential tag array lookups at the end-points. In figure 6 we report normalized dynamic power dissipation for tag array lookups. Our evaluation shows a dynamic power reduction of 53.2% and 52.3% in tag arrays lookup is achievable when we use S-PTC.

Our evaluation shows that exploiting S-PTC increases static power in tag arrays (60% and 140% in 4-way and 8-way CMPs) as a result of the auxiliary structures used.

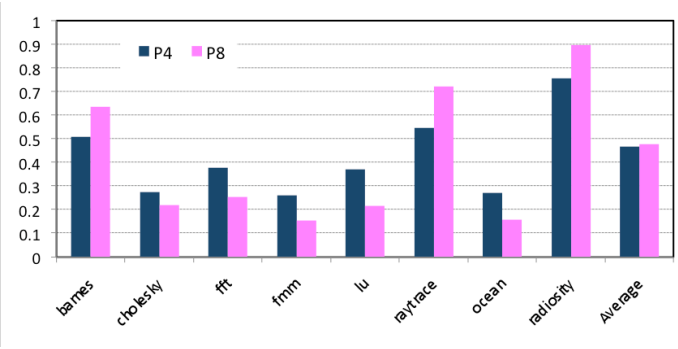

Figure 6: *Tag array dynamic power dissipation*

Static power increases with the number of cores but could be reduced by exploiting static power reduction solutions, which is part of our future work.

5.5 Area

We estimate chip area utilization using the number of bits stored. For 40-bit address space CMPs with 64B cache blocks, a complete tag array should contain 32 bits for the tag address, and valid and dirty bits for every individual block. On the other hand, the S-filter stores 10 bits (8 bit for LSB, one valid bit and a dirty bit) for each cache line. Therefore S-PTC stores 10 bits per-core for every 34-bit tag entry of bus-side tag arrays.

6. Related Work

A plethora of studies have investigated interconnect and memory optimizations in shared memory multiprocessors. Multicast snooping [14] and destination set prediction [15] aim at achieving a bandwidth requirement comparable to directory coherency while maintaining low access time similar to snoop based systems. Atoofian and Baniasadi showed that there is a high chance that two consecutive cache misses in a local cache are supplied by the same remote node (supplier locality). They exploited this to reduce power by limiting cache lookup and snoop broadcast to the predicted supplier [16]. Jetty is a destination-based snooping filter for symmetric multiprocessor systems, which takes two different approaches (i.e., inclusion and exclusion) to eliminate extra accesses to local L2 caches [17]. Ekman et al. evaluated Jetty for CMPs and reported that power savings achieved by jetty are often outweighed by the overhead associated with the filters [18]. In Blue Gene/P super computer three filters are used to optimize the memory system [6]. Ballapuram et al. use bloom filters [19] to remove unnecessary coherence activities [7]. Serial snooping is a destination-based non-speculative snooping technique that takes a sequential approach to search other cores instead of broadcasting [20]. Flexible snooping further improves serial snooping in logical ring interconnects by employing an adaptive filter on each core that decides to snoop the request and then forward or to snoop and forward in parallel [21].

Ekman et al. save sharing patterns for each memory page to filter unnecessary snoop requests. Two further studies [4,5] introduce region (as a contiguous power-of-two

number of cache lines) to address snoop inefficiencies. RegionScout showed that memory requests lead to global region misses frequently and used a non-tagged filter to keep track of region's sharing pattern [4]. Cantin et al. used a tagged structure to save information for more regions but at the cost of complexity [5]. Both of these works prevent snooping when they find other caches not sharing any block of the region containing the requested address. These studies do not improve bandwidth or memory utilization for cores not sharing the region when they attempt to broadcast. In-Network-Coherence-filtering addressed this issue by adding a table for each output port of on-chip routers containing non-shared regions of accessible cores [3].

In addition to the general differences listed above, our work is different from RegionScout [4] and Coarse-Grain Coherence Tracking [5] as S-PTC reduces interconnect and cache activity for shared regions too. While S-PTC limits interconnect and memory activity to those cores likely to have the data[1], region-based techniques broadcast to all nodes and caches under such circumstances. Previous studies relying on temporal locality in cache misses (e.g., [6, 17]) capture a superset of locally stored tags by means of counting bloom filter [7, 17]. This approach misses some saving opportunities as a result of aliasing. S-filters, on the other hand, are not vulnerable to aliasing as they save low resolution bits for every individual tag. Moreover, unlike [17] and [6], S-PTC can capture all kind of misses including those that have occurred very recently.

While serial snooping and flexible snooping have variable snoop delay, S-filters come with fixed penalty. Furthermore and in contrast to [20], S-PTC can improve performance as discussed earlier.

7. Conclusion

In this work we introduced S-PTC to improve energy efficiency in CMPs. S-PTC uses a low number of tag bits to reduce the amount of data processing and communication in CMPs. S-PTC reduces power as it eliminates a considerable share of the useless tag fast processing and transmission. We show that S-PTC reduces power while improving or maintaining performance for applications and configurations used in this study.

Acknowledgment

This work is supported by the Natural Sciences and Engineering Research Council of Canada, Discovery Grants Program and by Iran's Institute for Research in Fundamental Sciences (IPM).

[1] Assuming exploiting interconnect systems, which allow messages reaching destination(s) without using the entire interconnect resources (e.g., star-like interconnect).

References

[1] S. V. Adve and K. Gharachorloo. Shared Memory Consistency Models: A Tutorial. Computer, Vol. 29, No. 12 (Dec. 1996), pp. 66-76.

[2] IBM. Power4: http://www.research.ibm.com/power4.

[3] N. Agrawal, L-S. Peh, and N. K. Jha. In-Network Coherence Filtering: Snoop Coherence without Broadcast. In Proceedings of International Symposium on Microarchitecture, New York City, New York, Dec. 2009.

[4] A. Moshovos. RegionScout: Exploiting Coarse Grain Sharing in Snoop-Based Coherence. In Proceedings of International Symposium on Computer Architecture, Jun. 2005.

[5] J. F. Cantin, M. H. Lipasti, and J. E. Smith. Improving Multiprocessor Performance with Coarse-Grain Coherence Tracking. In Proceeding of the International Symposium on Computer Architecture, Jun. 2005.

[6] V. Salapura, M. Blumrich, and A. Gara. Design and Implementation of the Blue Gene/P Snoop Filter. In Proceedings of International Symposium on High Performance Computer Architecture, Feb. 2007.

[7] C. S. Ballapuram, A. Sharif, and H.-H. S. Lee. Exploiting Access Semantics and Program Behavior to Reduce Snoop Power in Chip Multiprocessors. In Proceeding of the International Conference on Architectural Support for Programming Languages and Operating Systems, Mar. 2008.

[8] R. Kumar, V. Zyuban, and D. Tullsen. Interconnections in Multi-core Architectures: Understanding Mechanisms, Overheads and Scaling. In ***ISCA***, Jun. 2005.

[9] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In International Symposium on Computer Architecture, pages 24–36, Santa Margherita Ligure, Italy, Jun. 1995.

[10] University of Illinois at Urbana-Champaign. http://sesc.sourceforge.net, 2005.

[11] Sun Niagara. http://www.sun.com/processors/throughput/.

[12] N. Muralimanohar, R. Balasubramonian, and N. Jouppi. Optimizing NUCA Organizations and Wiring Alternatives for Large Caches with CACTI 6.0. In Proceedings of the 40th International Symposium on Microarchitecture, Dec. 2007.

[13] L. Cheng et al. .Interconnect-Aware Coherence Protocols for Chip Multiprocessors. In Proceeding 33rd International Symposium on Computer Architecture, IEEE CS Press, 2006, pp. 339-351.

[14] E. E. Bilir, R. M. Dickson, Y. Hu, M. Plakal, D. J. Sorin, M. D. Hill, and D. A. Wood. Multicast Snooping: A New Coherence Method using a Multicast Address Network. SIGARCH Computer Architecture News, pages 294–304, 1999.

[15] 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 International Symposium on Computer Architecture, Jun. 2003.

[16] E. Atoofian and A. Baniasadi. Using Supplier Locality in Power-Aware Interconnects and Caches in Chip Multiprocessors. Journal of Systems Architecture Vol 54, No. 5 (October 2007) pp. 507-518.

[17] A. Moshovos, G. Memik, B. Falsafi, and A. Choudhary. Jetty: Filtering Snoops for Reduced Energy Consumption in SMP Servers. In Proceeding of the 7th International Symposium on High- Performance Computer Architecture, Jan. 2001.
[18] M. Ekman, F. Dahlgren, and P. Stenström. TLB and Snoop Energy-Reduction Using Virtual Caches for Low-Power Chip-Multiprocessors. In Proceeding of ACM International Symposium on Low Power Electronics and Design, Aug. 2002.
[19] B. H. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. Communication of the ACM 1970.
[20] 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.
[21] K. Strauss, X. Shen, and J .Torrellas. Flexible Snooping: Adaptive Forwarding and Filtering of Snoops in Embedded-Ring Multiprocessors. In International Symposium on Computer Architecture, Boston, MA, Jun. 2006.