

# Time-based Snoop Filtering in Chip Multiprocessors

Iman Faraji  
CE Department  
Amirkabir University of Technology  
Tehran, Iran  
i\_faraji@aut.ac.ir

Amirali Baniasadi  
ECE Department  
University of Victoria  
Victoria BC, Canada  
amirali@ece.uvic.ca

## Abstract

*Conventional snoop-based Chip Multiprocessors (CMP) maintain coherency by broadcasting snoop requests to all remote caches. It has been shown that many snoop requests fail to find data in remote caches and hence are redundantly broadcast, wasting both energy and bandwidth. We are motivated by observing that there are phases during program runtime when shared data is scarcely ever available.*

*We exploit this observation and propose two solutions to filter redundant snoops. First, we introduce Time-based Global Miss Prediction (TGM) to detect intervals when snoop requests made by all processors fail. We avoid snooping during such intervals. Second, we introduce Time-based Local Miss Prediction (TLM), in which recent snoop activities on each node are monitored to detect local data scarcity periods. The goal of TLM is to shutdown snooping for a single node temporarily during such periods.*

*Both of our methods target CMPs using simple coherence protocols and exploiting write-through caches. Our proposed methods come with negligible area and power overhead, as they require small auxiliary structures. We reduce memory energy (max: 18%) and snoop traffic (max: 93%). Meantime, for some applications, we improve performance (max: 3.8%) as a result of early cache miss detection.*

## 1. Introduction

Chip Multi-Processors (CMP) were proposed to address problems associated with conventional processors. Such problems include the increasing power demand and memory access latency. CMPs take advantage of running parallel threads on multiple processors [1, 2, 3] and are expected to become even more popular in the future [21, 22].

In order to achieve high performance, snooping is used in small and mid-size CMPs to implement coherency protocols. Conventional snooping takes an aggressive approach, broadcasting requests to all processors. This approach, while providing low cache-to-cache latency, comes with high energy and bandwidth demand.

It has been shown that a high portion of snoop requests fail to find any data in remote  $L_1$  caches and hence are redundantly broadcast. Redundant snoops result in unnecessary energy and bandwidth consumption. In this work our goal is to eliminate such redundant snoops without compromising the overall performance. We are motivated by our observation that there are phases during program runtime, in which shared data is scarcely ever available. During such phases, snoop requests initiated by one or more processors fail consecutively. We exploit our

observation and propose two solutions to filter redundant snoops.

First, we introduce Time-based Global Miss prediction (TGM) to detect intervals in which snoop requests made by all processors fail. To detect these intervals, TGM uses aggregate information about last snoop outcomes in all processors. Once such periods are detected, we stop snooping for processors to avoid further redundant snoops.

Second, we propose Time-based Local Miss Prediction (TLM). TLM tracks recent snoop outcomes in a single node to detect local data scarcity periods. During these periods TLM stops snooping temporarily.

CMPs use both write-through (WT) and write-back caches (WB). WB caches and the associated coherency protocols (e.g., MESI) are popular as they come with lower memory traffic. On the other hand, WT caches can be used in systems using simple coherency mechanisms. For example, Blue Gene/P [4] which uses an integrated first level cache (along with a multilevel cache) can also operate in a WT mode. WT-based configurations require frequent snoop invalidate messages and therefore have a high number of snoop transactions. Consequently, WT caches appear to be better targets for snoop filters [5].

While previous studies have mainly focused on WB-based CMPs, in this study we investigate WT-based CMPs.

In summary, we make the following contributions:

- We show that, there are phases during program runtime that shared data is unavailable. During such periods snoop requests fail consecutively. We suggest efficient mechanisms to detect such periods.
- We introduce TGM, as a simple and low-overhead time-based snoop filtering mechanism to avoid redundant snoops in WT-based CMPs. We propose two variations for TGM: TGM-First and TGM-Last. On average, TGM-First and TGM-Last reduce snoop traffic by 58% and 57% respectively.
- We introduce TLM as a time-based snoop filtering mechanism that utilizes local information about last snoop outcomes in WT-based CMPs. TLM detects local data scarcity periods to detect and avoid redundant snoop requests made by a single node. TLM reduces snoop traffic by 77%.
- By using TGM and TLM, we reduce energy consumption in memory by up to 17%. In addition, for most benchmarks both techniques do not impose any performance loss. Meantime, for some benchmarks we improve performance (up to 3.8%).

The rest of the paper is organized as follows: In Section 2, we discuss background. In Section 3, we present our motivation. We discuss our techniques in Section 4. We review methodology and results in Section 5. In Section 6, we discuss simulation results in more detail. We review related works in Section 7. Finally, in Section 8 we make concluding remarks.

## 2. Background

To reduce program runtime in a snoop-based CMP, every time a node does not find a data locally, it broadcasts the data address (along with the other required information) to other processors connected to the bus. Meantime, all bus-side caches snoop the transactions on the bus and respond to the request based on whether they have a valid copy of the requested data or not. This aggressive mechanism provides high performance since all processors observe snoop packets simultaneously.

To enforce coherency, snooping schemes use sets of protocol states. Accordingly, each node may change its state in response to local requests or those coming from remote processors.

Typical snooping schemes, although providing fast cache to cache transfer, are not energy efficient as quite often processors initiating snoop requests end up not finding the requested data [7, 8, 9, 10] in other cores, also referred to as a redundant snoop. A redundant snoop, while utilizing processor and interconnect resources, does not contribute to the overall performance.

Snoop filters can be used in both WT and WB configurations to avoid such redundant snoops. WT configurations, however, can benefit more from such filters due to two reasons. First, WB-based coherency protocols, such as MESI, inherently reduce a considerable portion of snooperable transactions as they limit invalidation messages to shared data [5]. Second, WB-based snoop filters cannot have a speculative nature; as in the event of inaccurate prediction there will be a possibility of accessing stale data. For example consider a scenario where a copy of the requested data in the modified state exists in a remote local cache and the L2 cache is not updated yet. Under these circumstances, filtering the snoop request mistakenly could result in reading the wrong data. On the other hand, WT-based snoop filters have less to worry about as the L2 cache is always up-to-date.

## 3. Motivation

Our study shows that redundant snoops are sometimes followed by consecutive redundant snoops resulting in intervals of data scarcity. During such intervals consecutive global read misses (GRMs) or local read misses (LRMs) occur resulting in wasted energy and bandwidth. A GRM occurs whenever the last snoop requests of all processors fail. In other words, a redundant snoop initiated by a given processor is considered to be a GRM, if the last snoop requests by all other processors have already failed. We report GRM frequency in Figure 1 (see Section 5 for more details). GRM frequency is the share of redundant snoops that occur after the last snoop request in all other nodes has failed to

find the data in remote caches. According to Figure 1, GRM frequency reaches a minimum of 33% (for Barnes) and a maximum of 99% (for Cholesky, FFT, and Volrend). In other words, in Barnes 33% of time all processors face a redundant snoop simultaneously. We conclude that a considerable portion of redundant snoops are GRMs.

In Figure 1, we also report LRM frequencies. LRMX frequency shows how often we witness at least X consecutive number of local misses. In order to show that LRMs happen consecutively, we report the percentage of local misses occurring during at least three (presented as LRM3) or seven (presented as LRM7) consecutive times.

In other words, LRM3 and LRM7 represent the frequency of local misses occurring during data scarcity periods of more than three and seven consecutive LRMs respectively (e.g. an LRM that follows five consecutive LRMs on a given processor is considered to be an LRM3, but not an LRM7). In the interest of space we only report for intervals of three and seven consecutive misses. Other number of consecutive misses show similar trends.

We refer to intervals of consecutive global misses as *Global Data Scarcity Periods* or *G-DSPs*. We refer to intervals of consecutive local misses as *Local Data Scarcity Periods* or *L-DSPs*.

To provide better understanding in Figure 2(a) we report the share of global misses occurring during different G-DSPs for the applications and processor configuration studied here. In Figure 2(b) we report LRM requests distribution among various L-DSPs. We also report the share of each processor in the overall number of LRM requests. We categorize GRM requests into six periods of data scarcity (G-DSP<sub>0</sub> to G-DSP<sub>5</sub>). Similarly, we divide LRM requests into six DSPs (L-DSP<sub>0</sub>-L-DSP<sub>5</sub>). DSP<sub>0</sub>, DSP<sub>1</sub>, DSP<sub>2</sub>, DSP<sub>3</sub>, DSP<sub>4</sub> and DSP<sub>5</sub> notations refer to intervals of one, two to 100, 101 to 1000, 1001 to 10000, 10001 to 100000, and over 100000 consecutive failed requests, respectively. As reported in Figure 2(a), on average 0.1%, 3.3%, 9%, 8.9%, 55.5% and 23.2% of the snoop misses take place during G-DSP<sub>0</sub>, G-DSP<sub>1</sub>, G-DSP<sub>2</sub>, G-DSP<sub>3</sub>, G-DSP<sub>4</sub>, and G-DSP<sub>5</sub>, respectively. As reported in Figure 2(b), on average 0.2%, 9%, 3.2%, 27.8%, 39.8% and 20% of the local misses take place during L-DSP<sub>0</sub>, L-DSP<sub>1</sub>, L-DSP<sub>2</sub>, L-DSP<sub>3</sub>, L-DSP<sub>4</sub> and L-DSP<sub>5</sub>, respectively.

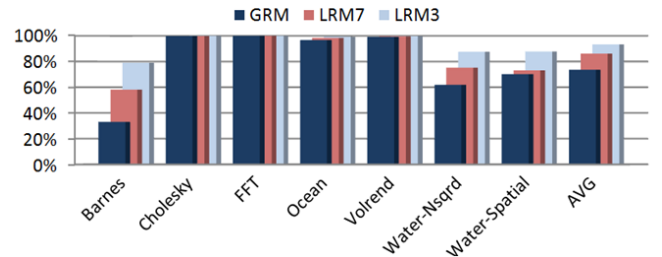


Figure 1: GRM and LRM frequency

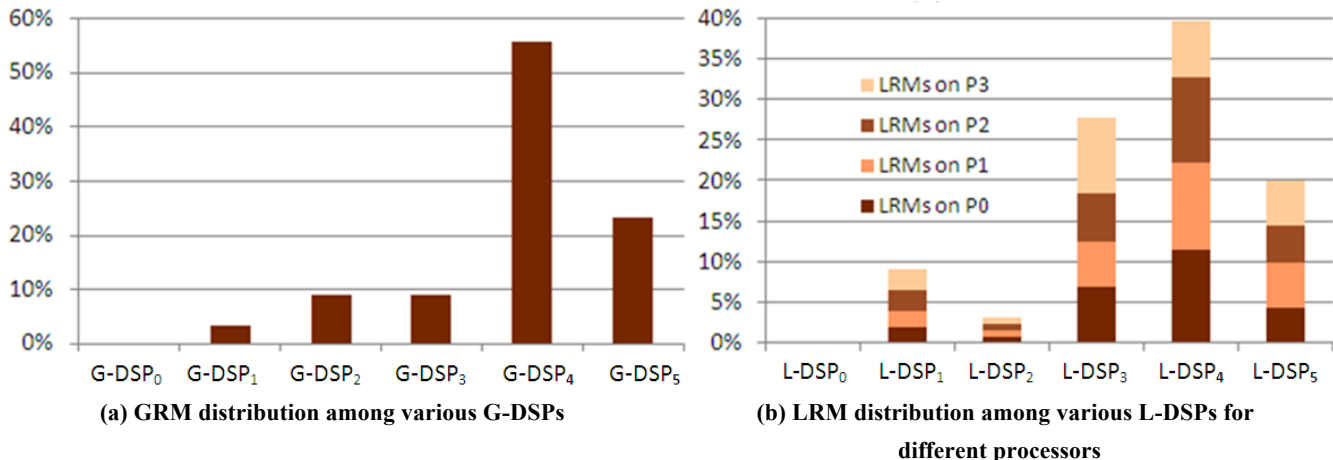


Figure 2: Distribution of redundant snoops among various DSPs

We conclude from Figure 2 that long period DSPs appear frequently during program runtimes. The energy and bandwidth exhausted during such periods does not contribute to performance. Therefore detecting DSPs and avoiding coherency activity (i.e., cache lookup and interconnect access) during them could improve energy and bandwidth efficiency without harming performance.

#### 4. Time-based Snoop Filtering

In this Section, we discuss our techniques. In Section 4.1, we discuss Time-based Global Miss Prediction (TGM). In Section 4.2, we review Time-based Local Miss Prediction (TLM). Both of our solutions target snoop-induced read misses. Considering the speculative nature of our solutions, we avoid stalling snooping and follow conventional snooping in the event of a write miss.

##### 4.1 Time-based Global Miss Prediction

###### 4.1.1 TGM

TGM builds on the observation that long periods of redundant snoops in all processors occur frequently resulting in wasted energy and activity. TGM aims at identifying long intervals where snooping of all processors fail. The goal is to shutdown snooping in as many processors as possible without compromising performance during such intervals.

TGM works as follows. When the last snoop request of all processors fail, snooping stops in all processors but one. While stopping snooping in all processors increases energy savings it can potentially degrade performance. To avoid performance loss we keep one processor snooping. We refer to this active processor as the surviving processor. Depending on which processor is selected as the surviving processor, TGM has two variations. In the first variation, TGM-First, we pick the first processor that has failed snooping as the surviving processor. In the second variation, TGM-Last, we pick the last processor that has failed snooping as the surviving processor.

Whenever TGM decides to stop snooping in processors, the next read miss request made by one of the impacted processors is treated as a GRM effectively avoiding accessing other local caches. Therefore, this request is directly sent to the lower level of memory, ignoring whether the requested data exists in remote local caches or not. If the requested data does not exist in any of

the remote local caches, the predicted GRM is an accurate one. Otherwise, the TGM prediction is inaccurate. This would not harm our WT-based system as no stale data is accessed in the event of inaccurate prediction.

In Figure 3, we report coverage and accuracy for TGM. We define coverage as the percentage of GRMs identified by TGM. Accuracy is defined as the percentage of GRM predictions that turns out to be correct. As reported in Figure 3(a), TGM-First and TGM-Last cover 63% and 63.2% of total GRMs respectively. According to Figure 3(b), the average accuracy of TGM-first and TGM-last is 93.5% and 94% respectively.

###### 4.1.2 Implementation

TGM uses the last snoop outcome for all processors to identify GRM periods. To record the last snoop outcome for each processor we add a single bit to each processor referred to as the Last Snoop Status (LSS) bit. LSS bit of a processor is set to one if the last snoop request made by that processor fails to return the data from other remote caches. LSS is reset to zero when the last snoop request made by the processor succeeds in finding the missing data in a remote cache.

Whenever LSS bits for all processors are set to one, the snoop disabling signal is generated. At this point, upon the next read miss request made by any processor (except the surviving processor which follows the conventional approach), the request is sent directly to the lower level memory. Meantime, if the surviving processor initiates a snoop request that hits in a remote local cache, all LSS bits are reset to zero and all processors restart broadcasting snoop requests again.

In our system, we use a Snoop Disable Logic (SDL) system. SDL decides whether to initiate snooping or skip this step and forward the request to the next level of memory. This decision is made based upon the snoop disabling signal generated by the predictor.

TGM requires  $n$  LSS bits, an  $n$ -input AND gate, and  $n$  SDLs where  $n$  is the number of processors in our CMP.

#### 4.2 Time-based Local Miss Prediction

###### 4.2.1 TLM

In TLM we use two separate counters; the first counter, which we refer to as the Redundant Snoop (RSN) counter, is responsible for counting LRMs. Each time a snoop request fails,

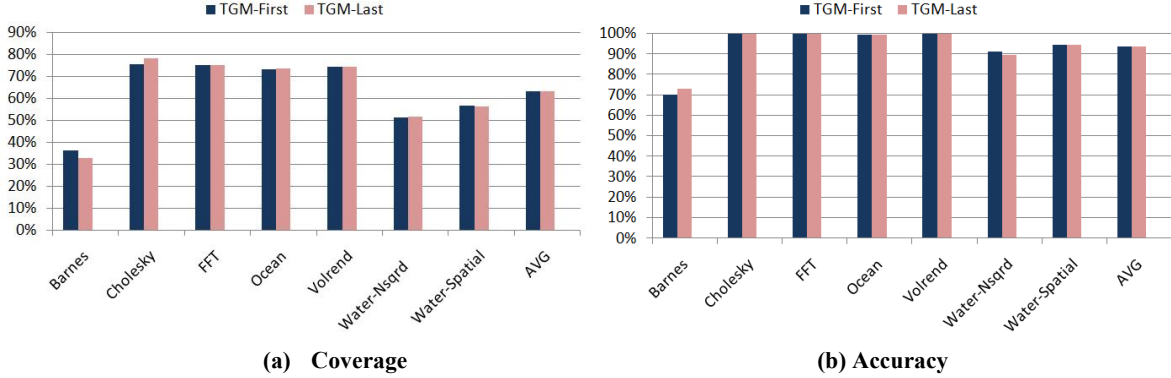


Figure 3: TGM coverage and accuracy

RSN counter increments, otherwise the counter value resets to zero. When RSN counter saturates, TLM generates and sends a snoop disabling signal to its own SDL. SDL directly sends a pre-decided number of upcoming read miss requests to the next memory level. A second counter, referred to as the Restart (RST) counter, sets this pre-decided number. The RST counter is responsible for counting the number of requests directly sent to the L<sub>2</sub> cache. Once the RST counter saturates, conventional snooping restarts. If the next snoop request finds the missing data in any of the remote caches, both the RSN and RST counters reset to zero, and conventional snooping continues.

In Figure 4, we show total LRM coverage and accuracy for TLM. We define coverage as the percentage of LRMs detected by TLM. Accuracy is defined as the percentage of LRM predictions which turn out to be an LRM. In this figure, we report for various RSN and RST counter sizes. Using large RSN counters leads to ignoring small consecutive LRM intervals. Consequently, when RSN counter size increases, LRM coverage decreases. The larger the counter size, the less number of times we disable conventional snooping. On the contrary, as the RST counter size increases, LRM coverage increases, while accuracy decreases. Using larger RST counters keeps SDL enabled for larger number of miss requests. This may lead to eliminating a higher number of snoop requests but not necessarily higher accuracy. By using larger RST counters, we may disable snooping for long periods. Since many DSPs are short, this could result in missing snooping opportunities, which could have resulted in a hit.

Counter size impacts LRM accuracy and coverage in two ways. First, in the event of detecting an LRM we forward the request to the L<sub>2</sub> cache memory. Inaccurate LRM predictions result in an unnecessary cost of high access time and energy consumption. As Figure 4 depicts, using large RSN counter and small RST counter reduces this cost. Second, higher coverage results in eliminating more redundant snoop activity. As we show in Figure 4, using small RSN counter and large RST counter makes this possible.

In the results reported in this paper we assume a 3-bit RSN counter (RSN-3) and a 4-bit RST counter (RST-4) as our study shows that this combination results in the highest energy delay product.

#### 4.2.2 Implementation

In a TLM-enhanced system each node is equipped with a predictor. This predictor is responsible for detecting LRMs and filtering them using the SDL.

To provide better understanding, we explain this technique assuming an x-bit RSN counter (RSN-x for short) and a y-bit RST counter (RST-y for short). The RSN counter is incremented if a snoop request fails; otherwise the counter resets to zero. When the RSN counter saturates (reaches  $2^x-1$ ), an SDL signal is generated. Upon the next read miss request, snooping does not take place, and the request is forwarded to the next level memory. Meantime, the RST counter is incremented. This scenario repeats for the upcoming read miss requests, until the RST counter saturates (reaches  $2^y-1$ ) and deactivates SDL signal for the next read miss request. Consequently, the processor follows the conventional snooping protocol. The request is broadcast on the interconnection requesting the missing data from remote caches. If the missing data exists in remote local cache, SDL signal remains deactivated and both RSN and RST counters reset to zero. Otherwise, if snooping fails, SDL signal is activated again, and only the RST counter resets to zero. Note that area and power overhead of this predictor is negligible, as each processor only includes two extra small saturating counters. Like TGM, TLM uses n SDLs. In addition, in TLM each processor uses a predictor that is equipped with two small (three and four bit) counters

## 5 Methodology and Results

### 5.1 Methodology

For our experiments, we use a subset of SPLASH-2 [17] benchmarks. We use three kernels (Barnes, Cholesky, and FFT) and four applications (Ocean, Volrend, Water-Nsqrd, Water-Spatial) each running for 500M of instructions after fast forwarding the first 500M instructions. We use and modify the SESC simulator [18] to simulate the CMP configuration used in this study. Our base system is the quad-core CMP presented in Table 1. We use 68 nm technology and assume a processor frequency of 5 GHz. We use Cacti6.5 to estimate the energy consumption for different cache-like structures [19].

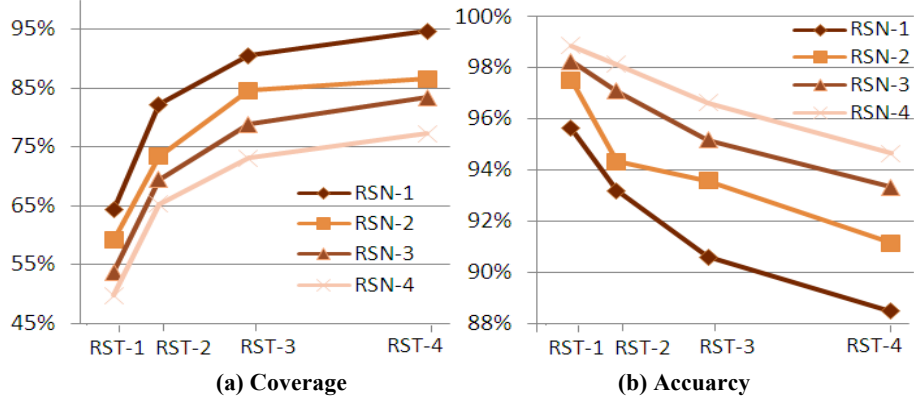


Figure 4: TLM Coverage and Accuracy using various RSN and RST counters

Table 1. SYSTEM PARAMETERS

| Processor   | Interconnect  | Memory   |
|---|---|--|
| <i>Frequency: 5 GHz</i><br><i>Technology: 68 nm</i><br><i>Branch Predictor: 16K entry</i><br><i>bimodal and gshare</i><br><i>Fetch/Issue/Commit 4/4/4</i><br><i>Branch Penalty : 17 cycles RAS: 32 entries</i><br><i>BTB: 2k Entries, 2 way</i> | <i>Data Interconnect: crossbar</i><br><br><i>Interconnect Width: 64 B</i> | <i>IL1: 32KB/ 2 way</i><br><i>DL1:32KB/2way/Write-Through</i><br><i>Access Time: 1 cycle</i><br><i>Block Size: 32</i><br><i>Cache line size: 32</i><br><br><i>L2:512KB/4way/Write-Through</i><br><i>Access Time: 11 cycles</i><br><i>Block Size: 32</i><br><br><i>Memory: 1GB</i><br><i>Access Time: 70 cycles</i> |

## 5.2 Results

In this Section we report snoop traffic reduction, memory energy reduction, average memory access latency and performance.

### 5.2.1 Snoop Traffic

In Figure 5 we report snoop traffic reduction. The left, the middle and the right bar for each benchmark show the fraction of snoop requests eliminated using TGM-First, TGM-Last and TLM respectively. TGM-First, TGM-Last and TLM reduce average snoop traffic by 58%, 57% and 77% respectively.

### 5.2.2 Memory Energy

Figure 6 shows the overall memory energy reduction compared to a conventional system. We assume memory energy to be the sum of the energies consumed in  $L_1$  local caches,  $L_2$  shared cache and the main memory. Memory energy reduction for TGM-First, TGM-Last and TLM is 8%, 8.5% and 11% respectively.

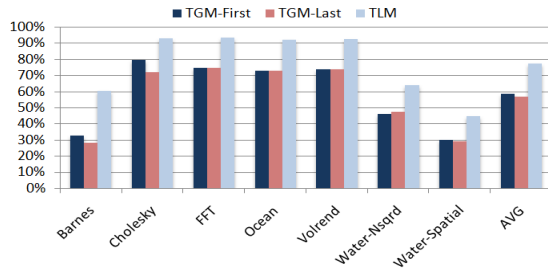


Figure 5: Relative snoop traffic reduction

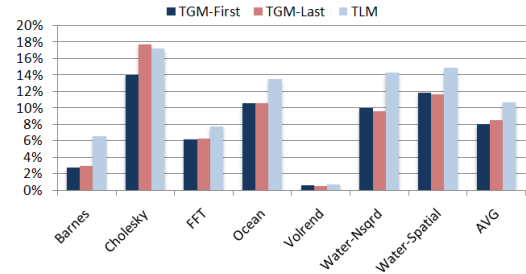


Figure 6: Relative memory energy saving

### 5.2.3 Average Memory Latency

In Figure 7 we report average memory latency compared to a system using a conventional cache snoop scheme. Average relative memory latency is the average time required to provide the missing data by using either remote caches, or the lower level memory ( $L_2$  or memory). TGM-First, TGM-Last and TLM have little impact on average latency. In all cases, but Barnes and Volrend, we even see a reduction in memory latency. On average TGM-First, TGM-Last, and TLM reduce memory latency by 1.1%, 2.1% and 1.7% respectively

### 5.2.4 Performance

In Figure 8 we report performance compared to the conventional system. Our solutions improve performance up to 3.8% for some benchmarks as the result of early memory access achieved by skipping redundant snooping. For some benchmarks (e.g., ocean) we witness negligible performance slowdown ( $\sim 1\%$ ). On average TGM-Last and TLM improve performance by 0.3% and 0.4% respectively. On average, TGM-First leads to almost no performance loss or improvement.

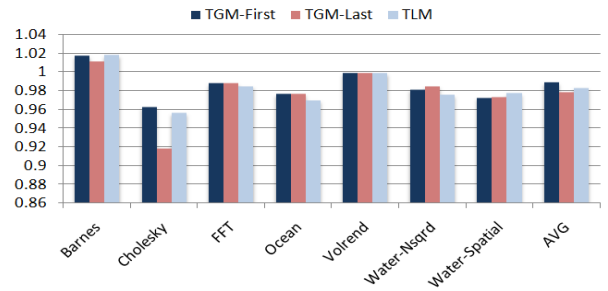


Figure 7: Relative average memory latency

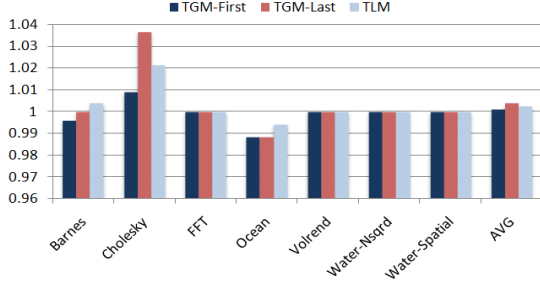


Figure 8: Relative Performance

## 6. Discussion

In Section 6.1 we discuss results in more detail. In Section 6.2 we compare memory energy to a WB-based CMP.

### 6.1 Analysis

In general, memory energy reduction and average memory latency depend on two parameters.

The first parameter is the fraction of redundant snoops in each benchmark. The higher the share of redundant snoops in a benchmark, the higher the potential of energy reduction and memory latency improvement. In Figure 9 we show the share of redundant snoops for the benchmarks used in our work. For each benchmark, this figure also shows the fraction of redundant snoops that hit (each bar’s upper part) and miss (each bar’s lower part) in the  $L_2$  cache memory.

The second parameter is the percentage of accurately detected redundant snoops by our solutions. While the first parameter indicates the potential in each benchmark, the second parameter decides the final improvement.

The benchmarks studied in this work could be categorized into three groups based on their memory energy reduction: The first group includes those benchmarks that show energy reductions higher than 15%. The only benchmark belonging to this category is Cholesky. Cholesky has a considerable share of redundant snoops. In addition, among all the benchmarks used in our experiments, it shows the highest accuracy and coverage under our solutions. Consequently, this benchmark shows high traffic, and memory energy and latency reduction.

The second group includes benchmarks showing moderate memory energy reduction (7% to 15%) (i.e. FFT, Ocean, Water-Nsqrd, and Water-Spatial). The results for this group are explained by the fact that they either have a low number of redundant snoops or, our solutions do not show high accuracy or coverage. For example, Water-Nsqrd and Water-Spatial show moderate accuracy. This results in unnecessary accesses to the lower level memory, negating the energy savings achieved by removing redundant snoops. Among all benchmarks, FFT (and to some extent Ocean) performs slightly different. FFT shows a high ratio of redundant snoops (indicated in Figure 9) and high traffic reduction (Figure 5). Also our solutions show a high accuracy and coverage for FFT (Figure 3). Despite these observations FFT shows low memory energy reduction (Figure 6). This is explained by the fact that FFT has a significant number of redundant snoop requests that not only miss in the  $L_1$  local caches but also do not find the requested data in the shared  $L_2$  cache (Figure 9). Hence the request is forwarded to the main memory. As main memory accesses consume high energy, the

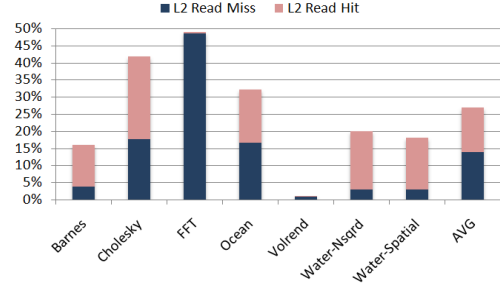


Figure 9: Share of redundant snoops

share of energy reductions achieved by eliminating redundant  $L_1$  accesses is reduced.

The third group includes benchmarks that show low energy reduction (less than 7%). We place Barnes and Volrend in this category. Among all benchmarks used in our experiments, Barnes has relatively low snoop reduction and shows the lowest coverage and accuracy. Using our solutions for this benchmark leads to low energy reduction and a slight increase in memory latency. However, while Volrend shows high snoop reduction, its energy savings is considerably low. This is explained by the low number of redundant snoops for this benchmark (Figure 9). Therefore, in spite of high accuracy and high snoop reduction the total number of eliminated snoops and consequently saved energy is low. Volrend comes with a low snoop frequency due to the following. This application renders a three-dimensional volume and shows good temporal locality. In this benchmark, the volume is represented as a cube of voxels (volume elements). Volrend renders several frames from changing viewpoints. Since consecutive frames in rotation sequences often vary slightly in viewpoint, much of the voxel data brought into the cache for rendering a given frame may still be available in the cache and reused on next frames, consequently reducing cache read misses and hence snoop frequency on new frames. In addition, Volrend provides good load distribution, improving spatial locality in local caches which results in further snoop frequency reduction [17, 20].

### 6.2 Energy in WB-based CMP

The snoop filters proposed in this paper target systems with WT caches. WB caches inherently reduce high number of snooper transactions by using sophisticated coherency protocols, and avoiding unnecessary write-backs. Figure 10 shows relative memory energy consumption if the WT cache used in our system is replaced with a WB cache. As reported, the WB configuration comes with lower memory energy consumption. Achieving this energy reduction requires using sophisticated coherency protocol.

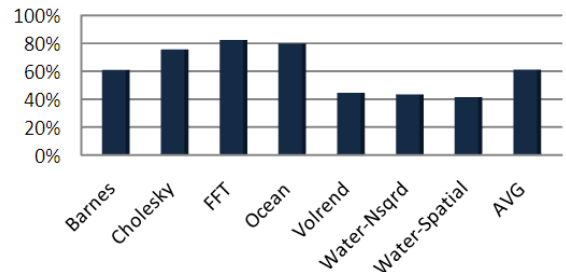


Figure 10: Relative memory energy consumption (replacing our WT cache with a similar size WB)

We conclude from this figure that CMP systems would save power by using WB caches equipped with complex coherency protocols. Meantime, if using simple coherency protocols comes with high priority then using WT caches exploiting snoop filters such as those suggested in this work improves energy efficiency.

## 7. Related Work

Previous studies have suggested several techniques to reduce snoop energy and traffic. Examples are snoop filters [7,8,9,10], speculative snoop [13] and serial snooping [14,15]. Other studies have suggested solutions to reduce energy in directory-based systems (e.g., Tagless Coherence Directory [16]). There are also studies aiming at reducing bandwidth consumption across different coherency protocols, e.g., CHOP [12].

Moshovos et al. [8] proposed Jetty as a snoop filter for symmetric multiprocessor (SMP) systems. Jetty is a small structure placed on the back side of  $L_2$  caches. Emkan et al. [16] demonstrated that jetty is not efficient in CMP configurations. In RegionScout [9] each node is equipped with a filter that is responsible to detect regions that do not exist in any remote caches. Chung and Kim [10] introduced broadcast filtering as a source-based snoop filtering mechanism. They used a snooping cache which stores all tag bits stored in all tag arrays. Agrawal et al. [7] introduced In-Network Coherence Filtering (INCF) to filter redundant snoops in order to save network bandwidth and power dissipation. Atoofian and Baniyadi used Selective Tag Lookup (STL) to avoid unnecessary cache accesses by filtering requests to the  $L_1$  cache that are likely to miss. They also introduced Speculative Selective Request (SSR) to reduce interconnect power by sending the request only to the speculated supplier node [13]. In Serial snooping [14] remote caches are checked serially instead of following conventional parallel snoop. Flexible snooping [15] enhances serial snooping by using an adaptive filter in each core. Zebchuk et al. [11] proposed Tagless Coherence Directory (TL) to address large energy and area overhead in conventional directory scheme.

Jiang et al. [12] introduced CHOP (Caching Hot Pages) for DRAM caching to increase memory bandwidth utilization. They suggested three types of filters: CHOP-FC, CHOP-MFC, and CHOP-AFC.

Our work is different from all above as we take a time-based approach deciding on snoop behavior changes during runtime. Unlike many previous solutions that save energy at the expense of performance, for some applications, we improve performance while reducing energy.

## 8. Conclusion

Our study showed that long periods of data scarcity exist during program runtime. In these periods, snoop requests fail to find the requested data in remote caches and hence are redundant. To address this issue we suggested two solutions for CMPs using WT caches. First, we introduced TGM, which uses global snoop behavior to detect and filter redundant snoops. Second, we proposed TLM, in which redundant snoops are filtered in local nodes. TLM counts recent redundant snoops to detect data scarcity periods and filter upcoming redundant snoops.

## References

[1] J. L. Hennessy and D. A. Patterson, "Computer Architecture: A Quantitative Approach", 3rd edition. San Francisco, CA: Morgan Kaufmann, 2003.

[2] D.W.Wall, "Limits of instruction-level parallelism", WRL Research Report 93/6, Digital Western Research Laboratory, Palo Alto, CA, 1993.

[3] B. A. Nayfeh, L. Hammond, and K. Olukotun, "Evaluating alternatives for a multiprocessor microprocessor," in Proceedings of 23rd Int. Symp. Computer Architecture, Philadelphia, PA, 1996.

[4] 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, February 2007.

[5] A. Ranganathan, "Experimental Analysis of Snoop Filters for MPSoC Embedded Systems", MASTER PROJECT, Ecole Polytechnique Federale de Lausanne, Switzerland, January 2010

[6] L. Hammond et al., "The Stanford Hydra CMP". IEEE MICRO Magazine, pages 71-84, March-April 2000.

[7] 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, NY, December. 2009.

[8] 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, January 2001.

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

[10] C. M. Chung and J. Kim. "Broadcast filtering: Snoop energy reduction in shared bus-based low-power MPSoCs". Journal of Systems Architecture - Embedded Systems Design, March 2009.

[11] J. Zebchuk, V. Srinivasan, M. K. Qureshi, and A. Moshovos. "A tagless coherence directory". In: MICRO 42: Proceedings of the 42<sup>nd</sup> Annual IEEE/ACM International Symposium on Microarchitecture, 2009.

[12] X. Jiang, N. Madan, L. Zhao et al, CHOP: "Adaptive Filter- based DRAM Caching for CMP Server Platforms", in: HPCA 15, High-Performance Computer Architecture, January 2010.

[13] E. Atoofian and A. Baniyadi. "Using Supplier Locality in Power-Aware Interconnects and Caches in Chip Multiprocessors". Journal of Systems Architectur, October 2007.

[14] C. Saldanha, M. Lipasti, "Power efficient cache coherence", in: Workshop on Memory Performance Issues (in conjunction with ISCA), June 2001.

[15] K. Strauss, X. Shen, J. Torrellas, "Flexible snooping: adaptive forwarding and filtering of snoops in embedded-ring multiprocessors", in: International Symposium on Computer Architecture, June 2006.

[16] M. Ekman, F. Dahlgren, and P. Stenstrom. "Evaluation of snoop-energy reduction techniques for chip-multiprocessors". In Proceedings of the First Workshop on Duplicating, Deconstructing, and Debunking(WDDD-1), May 2002.

[17] 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, Santa Margherita Ligure, Italy, Jun 1995.

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

[19] CACTI 6.5, <http://www.cs.utah.edu/~rajeev/cacti6.5/>, May 2010.

[20] J. Nieh and M. Levoy, "Volume Rendering on Scalable Shared-Memory MIMD Architectures", In Proceedings of the Boston Workshop on Volume Visualization, October 1992.

[21] J. Huh, D. Burger and W. Keckler. "Exploiting the design space of future CMPs", In Proc. 10th International Conference on Parallel Architecture and Compilation Techniques, September 2001.

[22] J. Held, J. Bautista, and S. Koehl. "From a few cores to many: A tera-scale computing research overview", Intel Research White Paper, 2006.