VGTS: Variable Granularity Transactional Snoop

Ehsan Atoofian

Electrical Engineering Department
Lakehead University
Thunder Bay, Canada
atoofian@lakeheadu.ca

Abstract. Transactional Memory (TM) is an appealing abstraction for increasing productivity of programmers and making parallel programming accessible to a wide community of non-experts. In TM systems, conflict detection is an essential element in maintaining correctness of transactions. Hardware Transactional Memories (HTMs) rely on cache coherence protocols to detect and resolve conflicts. In HTMs, when a transactional write misses in a cache, it broadcasts a snoop request asking remote caches for sharing information. While this method detects conflicts at the earliest possible time it is not efficient in term of power. We found that a significant fraction of transactional snoops in cache coherence protocols are unnecessary and waste power of interconnect and caches. Furthermore, many transactional snoops occur in coarse regions of memory. In this work, we introduce Variable Granularity Transactional Snoop (VGTS) which dynamically changes snoop granularity for transactions. VGTS monitors transactions and dynamically matches snoop granularity to the transactions' address patterns. Our simulation results reveal that VGTS is effective and reduces power of interconnect up to 44% and eliminates unnecessary cache snoops up to 43%.

Keywords: Hardware Transactional Memory, Synchronization, Cache Coherence Protocol, Power.

1 Introduction

Chip Multiprocessors (CMPs) are becoming mainstream in server, desktop, and even embedded systems to overcome power wall and other constrains in single-core processors. While CMPs reduce power and cost of cooling systems, they bring some unprecedented challenges. Software developers can no longer rely on next year's processor to hide the cost of new features in sequentially written software packages and gain speedup. In order to maintain pace of performance improvement in software applications, programmers need to develop parallel programs. The traditional method for parallel programming is lock. However, this approach entails difficult trade-offs: performance vs. complexity. Parallel programming with coarse-grain locking is simple but results in poor performance. On the other side, fine-grain locking yields better performance but is error-prone and difficult to understand and maintain.

F. Wolf, B. Mohr, and D. an Mey (Eds.): Euro-Par 2013, LNCS 8097, pp. 203–214, 2013.

Transactional Memory (TM) [1] is a promising programming model which addresses the problems of lock-based programming by providing the potential for performance of fine-grain locking with simplicity of coarse-grain locking. A transaction is a sequence of instructions that should execute atomically. In a TM program, a programmer just marks sections of the program as transactions and the underlying system guarantees atomicity. As such, in TM systems, programmers specify what should be done atomically, leaving the system determining how this is achieved. This relieves programmers from burden of worrying about synchronization bugs such as deadlock and convoying.

TM systems can be broadly classified into three categories: 1) Hardware Transactional Memory (HTM) [2], 2) Software Transactional Memory (STM) [14], or 3) Hybrid Transactional Memory (HyTM) [7]. STMs entail significant overhead and there is growing interest in transactional support in hardware to improve performance. HTM research has caught the attention of industry and some companies proposed extension for their architectures to support TM, such as Intel's Transactional Synchronization Extensions (TSX) [6], and the AMD Advanced Synchronization Facility (ASF) [15].

TM systems utilize resources in CMPs by execution multiple transactions concurrently. To maintain consistency of concurrent transactions, only conflict-free transactions commit successfully. A conflict happens when two or more number of transactions accesses a memory location and at least one of them writes into the memory location. In the event of conflict, only one transaction can commit and the rest should abort and restart or stall in the hope that the conflict is resolved later [2].

HTMs exploit cache coherence protocols to detect and resolve conflicts [2]. When a transaction writes to a memory address, it broadcasts a snoop request to invalidate cache blocks of other transactions that have accessed the same memory address. If a transaction finishes without having had any of its entries invalidated by remote caches, then the transaction commits by writing back its dirty entries in the cache (or write buffer) to the memory in lazy policy [17] or discarding its log buffer in eager policy [2]. On the other side, if another thread invalidates a transactional entry in the cache, the transaction aborts and restarts.

We find that cache coherence protocols dissipate a significant fraction of their power on unnecessary snoops when running transactional applications. Many transactions access disjoint memory regions and so a significant portion of transactional snoops result in cache misses. Therefore, the vast majority of the transactional snoops are unnecessary as they fail to find sharers in remote caches. In this work, we propose Variable Granularity Transactional Snoop (VGTS) to reduce unnecessary snoops generated by transactions. When a transaction executes, VGTS monitors transactional reads and writes and determines granularity of snoop requests generated by the transaction. Later, if the same transaction executes and broadcasts a snoop request, in addition to the requested address, we may ask remote cores to snoop a region of addresses. The responses received from remote cores are stored in a local filter and used for future snoops. In future, if a snoop request is found in the local filter, then the request is not broadcasted as none of the remote cores have the request. As such, VGTS avoids needless snoop broadcasts and reduces power of interconnection network and caches.

The rest of the paper is organized as follows. Section 2 provides background on how cache coherence protocols work in HTMs and motivates the need for adjusting snoop granularity per transaction. Section 3 discusses the implementation details and presents the hardware design of VGTS. Section 4 describes the evaluation methodology and presents quantitative results. Section 5 discusses related work and Section 6 concludes the paper.

2 Background

In this work, we use an HTM system similar to LogTM [2]. LogTM is an implementation of HTM which requires moderate augmentation of existing hardware and uses software support to restore state of a processor in the event of abort. LogTM implements eager version management by creating a per-thread transaction log in private caches, which holds the old values of all transactional writes. To detect transactional conflicts, LogTM uses a modified version of MOESI cache coherence protocol [10].

In LogTM, each cache block has two additional state bits: read and write bits. These bits are set if a transaction reads/writes a cache block. LogTM detects conflicts using MOESI and read/write bits. If a transactional read misses in a local cache, then the cache broadcasts a snoop request asking remote caches for the missed address. If the missed address exists in a cache of a remote node, then the remote node responds to the snoop request. If read bit of the remote cache block is set, then the two transactions can proceed without conflict. However, if the write bit of the remote cache block is set, then the two transactions conflict and at least one of them should abort. Similarly, if a transactional write results in a cache miss, then a snoop request is broadcasted on the interconnection network. If the address exists in a remote cache and the corresponding read or write bit is set, then a conflict happens and one of the two transactions should abort. To resolve the conflict, the transaction which initiates snoop aborts and the other transaction proceeds. In LogTM, a software handler walks through the transaction log and restores register and memory values in the event of abort.

LogTM broadcasts a snoop request to all cores on a cache miss. If all of the cores are caching the missed address, the broadcast would not be wasteful. However, in some applications, a significant part of snoop requests misses in remote caches. Figure 1 shows the percentage of transactional coherence requests that are redundant in Stamp v0.9.10 benchmark suite (please refer to Section 4.1 for methodology and configuration). A redundant coherence request is one that does not exist in any of the remote caches, thus unnecessarily wastes processor resources. We found that more than 75% of coherence requests are redundant across all benchmarks and thus unnecessarily consume network power, while also resulting in redundant snoop-induced cache look-ups.

3 Implementation

VGTS dynamically adjusts snoop granularity for transactions. When a transaction executes, VGTS monitors addresses generated by the transaction. If the transaction

accesses consecutive memory locations then VGTS sets the snoop granularity to the size of the region accessed by the transaction. If the transaction accesses multiple disjoint regions, then VGTS sets the snoop granularity to the size of the smallest region to avoid wasting network bandwidth. To provide better insight into snoop granularity found by VGTS, we discuss part of a code region taken from Labyrinth. Figure 2.a shows part of the Labyrinth program. This benchmark finds shortest paths between pairs of starting and ending points for a given maze. gridPtr is a pointer to the shortest paths found in this benchmark. Figure 2.b shows address of cache blocks generated by the program in 2.a. Note that the granularity of snoops in cache coherence protocols is cache block. So, VGTS analyses addresses in the granularity of cache blocks not bytes. The sequence of addresses in 2.b forms three disjoint regions. The size of the smallest region is two cache blocks (15, 16). So, VGTS sets granularity of the transaction in 2.a. to two.

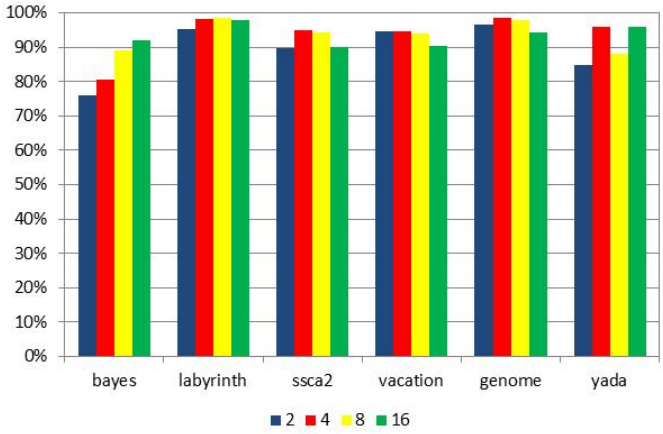


Fig. 1. Redundant snoops in Stamp v0.9.10 benchmark suite. In each benchmark, the number of threads changes between two and 16.

VGTS is a speculative approach and relies on the most recent execution of a transaction to decide on snoop granularity. VGTS is effective only if snoop granularity of a transaction does not change over time. To investigate pattern of snoop granularities in transactions, we measure locality of snoop granularity in Stamp benchmarks. Locality of snoop granularity shows how often snoop granularity of a transaction is the same in two consecutive executions. Figure 3 shows locality of snoop granularity in Stamp benchmarks [7]. For each benchmark, the number of threads varies from two to 16. The locality is measured by counting the number of times snoop granularity is the same in two consecutive executions of transactions and dividing by the total number of transactional commits. The locality changes from 63% in Labyrinth to 99.9% in Ssca2. This Figure proves that snoop granularity of transactions is predictable.

```
TM_BEGIN();
        …
        TMGRID_ADDPATH(gridPtr, …);
        …
TM_END();
```

a)

address
8, 9, 8, 10,11
15, 16
21, 22, 22,
23, 24
…

b)

Fig. 2. a) Part of the Labyrinth program. b) Memory addresses (in granularity of cache block) pointed by gridPtr in Labyrinth.

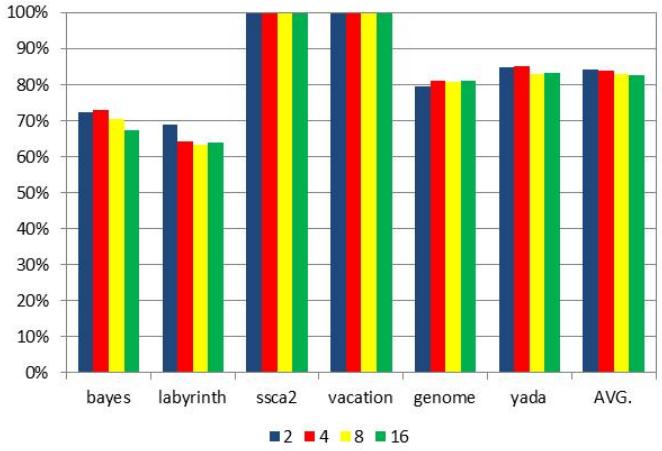


Fig. 3. Locality of snoop granularity in Stamp v0.9.10 benchmarks. For each benchmark, the number of threads changes from two to 16.

VGTS relies on Snoop Granularity Tables (SGTs) to keep record of snoop granularities of transactions (Figure 4). Each core has its own SGT. Each entry in SGT comprises two fields: snoop granularity and valid bit. When a transaction commits, VGTS writes snoop granularity of the transaction in to SGT. When a transaction starts, VGTS indexes SGT using starting address of the transaction. If a valid entry exists, then VGTS uses the corresponding snoop granularity for the transaction.

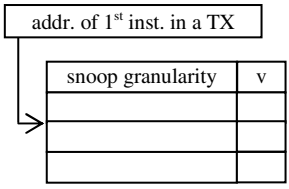


Fig. 4. Snoop Granularity Table (SGT)

VGTS exploits filters to reduce redundant coherence requests. The filter sits alongside the last-level private cache and maintains local region-level sharing information. Figure 5 shows the structure of the filter. The filter is a simple table with entries comprising the starting block address of a region, size of the region, and a valid bit. If a transactional instruction experiences a cache miss for the first time, then the cache controller broadcasts a snoop request asking other nodes to provide sharing information for the region determined by SGT. If the region is not shared by any other nodes, then the cache controller allocates an entry in the filter and sets block address, size, and valid bit. Prior to issuing a snoop request, each node looks up the local filter and if a matching entry exists, then the cache controller knows that forwarding the request is redundant. The filter entries are evicted either as a result of limited space or when the cache controller receives a snoop request for a matching region.

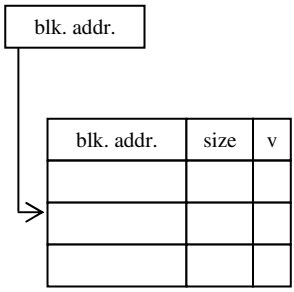


Fig. 5. Structure of a filter

4 Evaluation

In this section we evaluate VGTS. We describe our simulation environment in section 4.1. Then, in section 4.2, we present results for VGTS.

4.1 Experimental Setup

For all our evaluations, we perform full-system simulation using Gem5 [8].We model a CMP based on Alpha 21264 architecture. Each core has a private instruction cache and data cache. Table 1 shows configuration of the processor. We used Orion 2.0 [9] to estimate power consumption in bus. We also model the extra region links and the power consumed by these links. To estimate the overhead of filters, we used CACTI [16]. The technology node that we assumed is 65 nm.

We use Stamp v0.9.10 benchmark suite [7] to evaluate VGTS. We evaluated VGTS against baseline scheme. To find out the appropriate filter size, we explored a design with infinite table entries (Oracle) and four designs with 16, 32, 64, and128 entries, respectively.

We use an 8-entry SGT in our evaluations. We found that aliasing in a tag-less SGT with 8-entry is close to zero and increasing the size of SGT beyond 8 does not improve VGTS further.

Table 1. Configuration of Processors Modeled by Gem5 Full System Simulator

Benchmarks	Input Parameters
Processors	2-16 cores Alpha ISA, 2GHz
L_1 I&D Caches	64kB, 2-way associative, 64-byte line size, 1 cycle latency
Interconnect	Shared bus running at 1GHz
Orion Parameters	65 nm, V_{dd}: 1-V
L_2 Cache	Shared 2MB, 8-way associative, 64-byte line size, 10 cycles latency
Main Memory	2048MB, 100 cycles latency

4.2 Results

Figures 6 shows the normalized bus power (smaller is better) in VGTS with filter sizes being infinite (Oracle), 16, 32, 64, and 128 entries. The number of threads in benchmarks is eight. All values are normalized to bus power in the baseline scheme. By filtering redundant snoops, VGTS is able to reduce bus power. The figure shows that on average, VGTS reduces bus power by 14%, 16%, 17%, and 22% for filter sizes of 16, 32, 64, and 128, respectively. It can also be seen that even with an infinite filter table size, the bus power can be reduced by 24%, on average. This shows that VGTS with 128 entries filters is reasonably close to what an Oracle filter implementation would achieve.

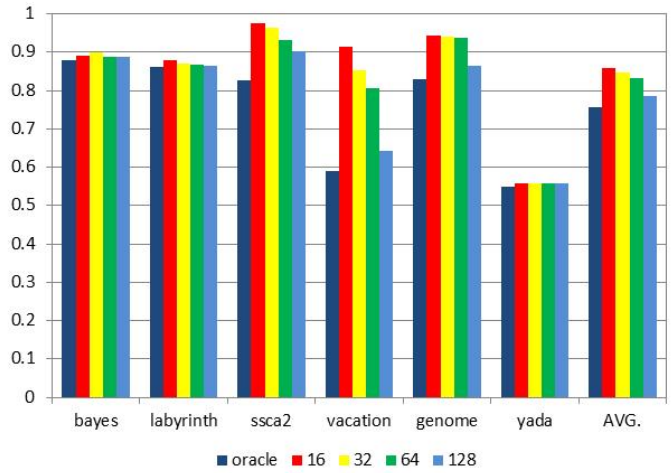


Fig. 6. Bus power reduction in VGTS relative to the baseline scheme when the number of threads is 8. For each benchmark, we use filters with infinite, 16, 32, 64, and 128 entries.

Figure 7 shows energy of snoop induced tag lookups in VGTS relative to the baseline scheme (smaller is better) when the number of threads is eight. The energy overhead of filters is included in Figure 7. In a CMP, both the processor and the bus access the L1 cache. With only one tag array, if both processor and bus need to access tag array simultaneously, one side has to stall which will degrade the overall performance. To address this problem we use two mirror tag arrays [10], one for the processor side and one for the bus side. Figure 7 shows energy reduction in the tag array on the bus side. On average, VGTS reduces energy of snoop induced tag lookups from 14% to 21% when the filter size varies between 16 and 128. An oracle filter eliminates snoop induced tag lookups by 25% which is close to a filter with 128 entries.

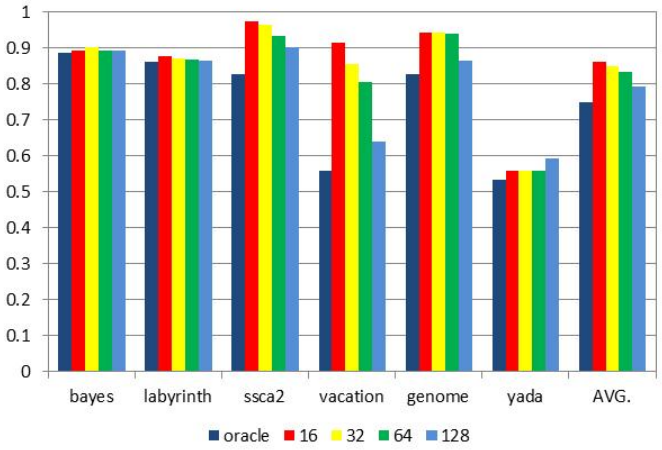


Fig. 7. Energy of snoop induced tag lookups avoided by VGTS when the number of threads is 8. For each benchmark, we use filters with infinite, 16, 32, 64, and 128 entries.

To study the scalability of our filtering proposal, we performed the above experiments for 2, 4, 8, and 16 threads. All parameters, as shown in Table 1, remain the same except the number of cores. The number of cores increases and is equal to the number of threads. Figure 8 shows the normalized bus power consumption (smaller is better) in the system with filter size equal to 64. In most of the benchmarks, power of bus changes negligibly with the number of threads. This shows that VGTS is able to maintain its power efficiency across different number of cores.

To provide better insight, we compare VGTS with RegionScout [11]. RegionScout [11] exploits coarse-grain data sharing information to reduce power of bus and snoop-induced tag lookups. In RegionScout, memory is statically divided into a number of regions. The hardware keeps record of non-shared regions and avoids broadcasting unnecessary snoops. The main limitation of RegionScout is that region size is determined statically. On the other side, VGTS adjusts snoop granularity dynamically and based on applications' behavior. Figure 9 shows power of bus and number of snoop-induced tag lookups in RegionScout. We change region size from 2KB to 16KB [11]. The number of threads is 8 and filter size is 128-entry in both RegionScout and VGTS. While RegionScout improves power of bus (Figure 9.a.) it increases

snoop-induced tag lookups significantly (Figure 9.b.). In Labyrinth, RegionScout increases snoop-induced tag lookups up to 461 times. The main reason that RegionScout falls behind VGTS is that RegionScout decides on snoop granularity once and uses it across all applications. However, VGTS dynamically changes snoop granularity based on pattern of addresses generated by transactions and is able to reduce both power of bus and snoop-induced tag lookups.

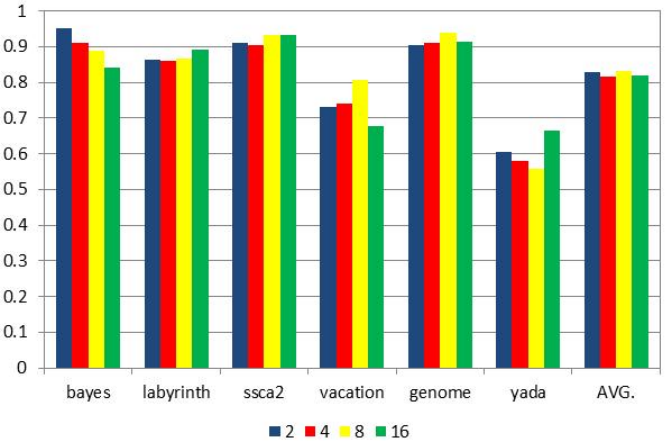
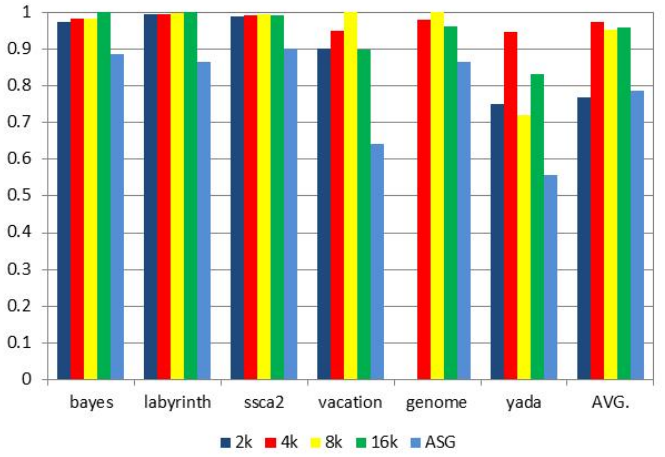


Fig. 8. Power of bus in VGTS relative to the baseline scheme. For each benchmark, the number of threads varies between two and 16.

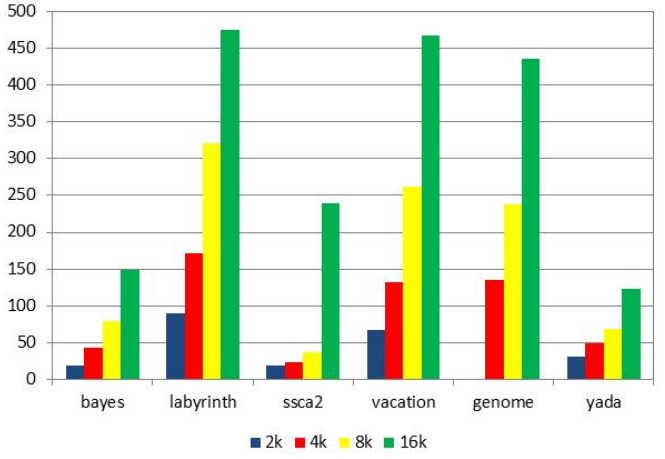
In the interest of space, we do not show a Figure for runtime in VGTS. VGTS reduces runtime of Stamp applications by 3.1% on average with maximum slowdown of 0.4%.

5 Related Work

Ferri et al. [19] proposed an energy efficient transactional memory design for embedded processors. They used a fully-associative Transactional Cache (TC) to hold speculative values generated in transactional sections. In the event of overflow in TC, the system enters an inefficient serial mode to allow the transactions complete. To reduce power consumption of memory hierarchy, they embedded a new mechanism in the memory hierarchy. When the mechanism is triggered, the contents of the TC are flushed into the traditional cache hierarchy allowing TC to power down. In a subsequent work, Ferri et al. [18] investigated alternative cache architecture for transactional values. This architecture aims at reducing the likelihood of cache overflow with the additional goal of reducing overall energy consumption. L1 cache is the primary storage space for holding both transactional data and non-transactional data. In addition, a small victim cache holds transactional data evicted from the L1 cache due to conflict misses. They show that the victim cache reduces pressure from sharing and improves energy-delay product.



a)



b)

Fig. 9. a) Bus power in RegionScout and VGTS relative to the baseline scheme. b) Snoop induced tag lookups in RegionScout relative to the baseline scheme. In both RegionScout and VGTS, number of threads is 8 and size of filters is 128-entry.

Gaona-Ramirez et al. [4] compared performance and energy of two well-known HTM systems: LogTM-SE [3] and TCC [17]. These two HTMs employ opposite policies for data versioning and conflict management. LogTM-SE uses eager policy and TCC use lazy policy for data versioning and conflict management. Gaona-Ramirez et al. showed that although on average TCC beats LogTM-SE, there are considerable deviations in performance depending on the particular characteristics of each application. Contention in LogTM-SE results in a large number of either stalled or aborted transactions depending on their write sets interactions. This behavior increases network traffic due to the persistent stall process. On the other side, TCC guarantees that at least one transaction will be able to commit in the presence of contention.

VGTS is different from all the above work since it focuses on cache coherence protocol to improve power consumption in HTMs.

A number of prior mechanisms relied on the tendency of coherence requests to reduce power in shared memory multiprocessors. Moshovos et al. proposed Jetty [12] to reduce power consumption in L2 cache. They exploited a filter between L2 cache and bus to skip snoops that would miss in L2 caches. VGTS is different since it focuses on regions of consecutive addresses to optimize power of interconnect and caches.

Ballapuram et al. [13] exploited the semantics of variables in a program to optimize snoops in a shared memory environment. Stacks are local to threads and they are not visible to the outside world. Conventional cache coherence protocols do not differentiate private and shared data. Ballapuram et al. proposed two techniques to eliminate snoop probes for stacks: Selective Snoop Probe (SSP) and Essential Snoop Probe (ESP). SSP is implemented in hardware and ESP is implemented in compiler. The advantage of using the SSP is that previously compiled binaries can benefit from this technique without recompilation. However, as there is no information provided by the software, the energy savings achieved is limited. On the other hand, the ESP technique lets the compiler takes full advantage of programs semantics to achieve higher energy savings. SSP and ESP can be combined with VGTS to reduce snoop power associate with stack regions.

Lotfi-Kamran et al. [5] investigated power of coherence directories when running commercial server and scientific workloads. They found that a significant fraction of directory power is dissipated on unnecessary lookups. They proposed TurboTag, a filtering mechanism to skip unnecessary directory lookups. VGTS is different since a requesting node can determine in advance that a request would miss in all the other nodes. With TurboTag, every node still broadcasts snoop requests to remote nodes. Advance knowledge of global region misses allows optimization of bus power that is impossible with TurboTag.

6 Conclusion

In this paper, we proposed and evaluated VGTS which is a novel snoop filtering mechanism for HTMs. We observed that many transactional snoop requests not only do not hit in any remote caches but also do not find any other blocks in a much larger surrounding region. VGTS is a speculative approach and monitors transactional reads and writes to decide on snoop granularity. VGTS exploits a set of filters to keep track of non-shared regions. These filters are transparent to the programmers and do not impose any limits on content of caches. We evaluated the design of VGTS in a full-system simulator and found that VGTS reduces power of bus and avoids many unnecessary cache snoops.

References

1. Herlihy, M., Moss, J.E.B.: Transactional memory: Architectural support for lock-free data structures. In: The Proceedings of the Twentieth Annual International Symposium on Computer Architecture (1993)

2. Moore, K.E., Bobba, J., Moravan, M.J., Hill, M.D., Wood, D.A.: LogTM: Log-based transactional memory. In: The Proceedings of HPCA, pp. 254–265 (2006)
3. Yen, L., Bobba, J., Marty, M.R., Moore, K.E., Volos, H., Hill, M.D., Swift, M.M., Wood, D.A.: LogTM-SE: Decoupling hardware transactional memory from caches. In: HPCA-13, pp. 261–272 (February 2007)
4. Gaona-Ramírez, E., Gil, J.R.T., Fernández, J., Acacio, M.E.: Characterizing Energy Consumption in Hardware Transactional Memory Systems. In: The Proceedings of SBAC-PAD 2010, pp. 9–16 (2010)
5. Lotfi-Kamran, P., Ferdman, M., Crisan, D., Falsafi, B.: TurboTag: lookup filtering to reduce coherence directory power. In: The Proceedings of ISLPED, pp. 377–382 (2010)
6. Intel Corp. Intel Architecture Instruction Set Extensions Programming Reference, 319433-012a edition (February 2012)
7. Minh, C.C., Trautmann, M., Chung, J.W., McDonald, A., Bronson, N., Casper, J., Kozyrakis, C., Olukotun, K.: An Effective Hybrid Transactional Memory System with Strong Isolation Guarantees. In: The Proceedings of ISCA (June 2007)
8. Binkert, N., Dreslinski, R., Hsu, L., Lim, K., Saidi, A., Reinhardt, S.: The M5 simulator: Modeling networked systems. IEEE Micro 26(4), 52–60 (2006)
9. Kahng, A.B., Li, B., Peh, L., Samadi, K.: Orion 2.0: A fast and accurate NoC power and area model for early-stage design space exploration. In: The Proceedings of DATE (2009)
10. Culler, D.E., Singh, J., Gupta, A.: Parallel Computer Architecture: A Hardware/Software Approach. Morgan Kaufman Publishers, San Francisco (1999)
11. Moshovos, A.: RegionScout: Exploiting Coarse Grain Sharing in Snoop-Based Coherence. In: The Proceedings of ISCA, Washington, DC, USA (2005)
12. Moshovos, A., Memik, G., Falsafi, B., Choudhary, A.: JETTY: Filtering Snoops for Reduced Energy Consumption in SMP Servers. In: The Proceedings of HPCA (2001)
13. Ballapuram, C.S., Sharif, A., Lee, H.S.: Exploiting access semantics and program behavior to reduce snoop power in chip multiprocessors. In: The Proceedings of ASPLOS (2008)
14. Dice, D., Shalev, O., Shavit, N.: Transactional Locking II. In: The Proceedings of the 20th International Symposium on Distributed Computing, pp. 194–208 (September 2006)
15. Chung, J., Yen, L., Diestelhorst, S., Pohlack, M., Hohmuth, M., Grossman, D., Christie, D.: ASF: AMD64 Extension for Lock-free Data Structures and Transactional Memory. In: Proceedings of MICRO, Atlanta, Ga (December 2010)
16. Muralimanohar, N., Balasubramonian, R., Jouppi, N.P.: CACTI 6.0: A Tool to Model Large Caches, Technical report, Hewlett Packard (2009)
17. Casper, H.C.J., Carlstrom, B.D., McDonald, A., Minh, C.C., Baek, W., Kozyrakis, C., Olukotun, K.: A scalable, non-blocking approach to transactional memory. In: HPCA-13, pp. 97–108 (February 2007)
18. Ferri, C., Wood, S., Moreshet, T., Bahar, R.I., Herlihy, M.: Energy and Throughput Efficient Transactional Memory for Embedded Multicore Systems. In: The Proceedings of HiPEAC 2010, Pisa, Italy, January 25-27 (2010)
19. Ferri, C., Viescas, A., Moreshet, T., Bahar, R.I., Herlihy, M.: Energy Implications of Transactional Memory for Embedded Architectures. In: Workshop on Exploiting Parallelism with Transactional Memory and other Hardware Assisted Methods, EPHAM 2008 (April 2008)