Appears in the proceedings of the
16th International Symposium on Low Power Electronics and Design (ISLPED'10)

TurboTag: Lookup Filtering to Reduce Coherence Directory Power

Pejman Lotfi-Kamran† Michael Ferdman‡† Daniel Crisan† Babak Falsafi†

†Parallel Systems Architecture Lab
École Polytechnique Fédérale de Lausanne
http://parsa.epfl.ch

‡Computer Architecture Lab
Carnegie Mellon University
http://www.ece.cmu.edu/CALCM

ABSTRACT

On-chip coherence directories of today's multi-core systems are not energy efficient. Coherence directories dissipate a significant fraction of their power on unnecessary lookups when running commercial server and scientific workloads. These workloads have large working sets that are beyond the reach of on-chip caches of modern processors. Limited to capturing a small part of the working set, private caches retain cache blocks only for a short period of time before replacing them with new blocks. Moreover, coherence enforcement is a known performance bottleneck of multi-threaded software, hence data-sharing in optimized high-performance software is minimal. Consequently, the majority of the accesses to the coherence directory find no sharers in the directory because the data are not available in the on-chip private caches, effectively wasting power on the coherence checks. To improve energy-efficiency for future many-core systems, we propose TurboTag, a filtering mechanism to eliminate needless directory lookups. We analyze full-system traces of server and scientific workloads and find that over 69% of accesses to the directory find no sharers and can be entirely avoided. Taking advantage of this behavior, TurboTag achieves a 58% reduction in the directory's dynamic power consumption.

Categories and Subject Descriptors:

C.1.0 [**Computer Systems Organization**]: Processor Architectures - *General*

General Terms:

Design, Performance

Keywords:

Low Power, Coherence, Directory, Bloom, Filter

1. INTRODUCTION

Technological improvements in transistor voltage scaling slowed down significantly in the recent years, forcing power and energy to become the main constraints in processor design. To continue improving processor performance, the architecture should be modified to integrate multiple cores on-chip; with nearly all components re-designed for scalability and energy efficiency.

The coherence directory is a component in modern CMPs which is responsible for maintaining the information about the sharers of a cache block. To enforce cache coherence, an energy-consuming directory lookup must be performed for every miss in every private cache. As the number of on-chip cores increases, both the number of directory lookups and the energy consumed per lookup increase, resulting in a super-linear increase in energy with respect to core count. Today's manufacturers are already faced with a coherence directory power problem [13]. Furthermore, a fundamental redesign to solve the coherence directory power inefficiency is not expected in the near future. Therefore, architects need effective ways to reduce the power consumption of the existing on-chip directory structures to enable increased core counts for several technology generations.

We find that coherence directories dissipate a significant fraction of their power on unnecessary lookups when running commercial server and scientific workloads. These workloads typically have a small primary working set that can be captured in a private cache (32 or 64 KB) and a large secondary working set that does not fit in on-chip caches of modern processors [8]. Due to the large secondary working set, the workloads experience many private-cache misses, causing blocks to be frequently replaced, and rarely allowing them to remain in the private cache long enough to be accessed by another core. Therefore, the vast majority of the lookups in the on-chip coherence directory are unnecessary because they fail to find a sharer in the directory. Furthermore, in the cases where blocks are shared, highly optimized code tends to avoid active sharing and frequent coherence activity. Consequently, active sharing is limited in today's workloads and the majority of private cache misses lead to a needless lookup in the directory because there are no sharers.

In this work, we propose TurboTag, a filtering mechanism for the coherence directory to eliminate needless lookups. TurboTag is based on the Bloom filter [3], an imprecise but space-efficient data structure that tests set membership. TurboTag filters unnecessary directory accesses using a compact structure that keeps track of block addresses that are stored in the directory. We use trace-driven simulation of a multi-core system running server and scientific workloads to evaluate the TurboTag design. We find that TurboTag is highly effective at eliminating power-hungry directory lookups, eliminating 69% of all directory accesses and achieving over 58% reduction in the dynamic power consumption compared to a directory organization without TurboTag.

The rest of this paper is organized as follows. Section 2 provides the background on CMP coherence, Section 3 evaluates the oppor-

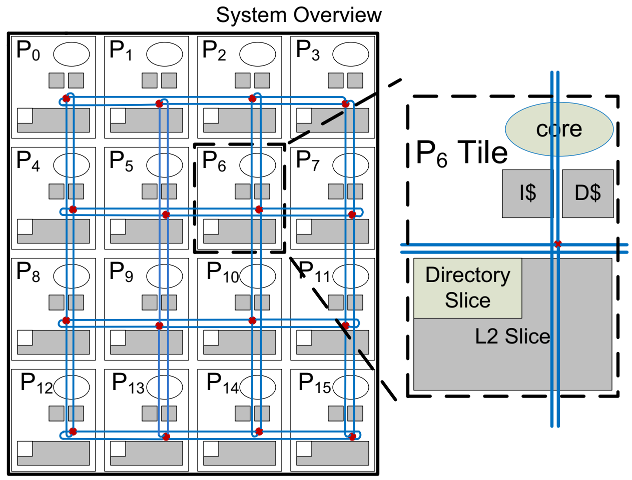

FIGURE 1. Example of a CMP. Upper-level caches are private to processor cores. Private-cache coherence is enforced with a directory structure, located next to the lower-level shared cache.

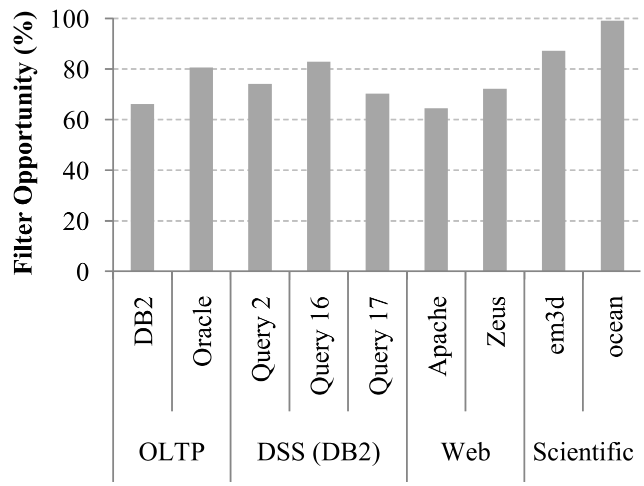

FIGURE 2. Filtering opportunity. Percentage of coherence directory data accesses that do not find the searched block.

tunity to reduce power consumption, and Section 4 discusses the design of TurboTag and its hardware implementation. We present the evaluation results in Section 5. Section 6 presents the related work. We conclude in Section 7.

2. DIRECTORIES IN CMPS

In this work, we consider a tiled CMP where many tiles are connected in an on-chip interconnect that shuttles messages between the tiles. Each tile in the system contains a processing core, a private cache, and a slice of the shared last-level cache as shown in Figure 1. Along with the shared cache slice, each tile collocates a slice of the coherence directory that keeps track of blocks maintained in the private caches. The memory address space is statically-interleaved among the L2 and directory slices.

Directory organizations can be roughly classified as duplicate-tag [2] or sparse [7]. A duplicate-tag directory is a cache that keeps in each of its sets a copy of the tag bits of all blocks residing in the sets of the private caches with the same index bits. Consequently, every slice of a duplicate-tag directory needs a number of sets equal to the number of sets of a private cache and an associativity equal to the aggregate associativity of all private caches (cores times private cache associativity). Because a duplicate-tag directory only stores the necessary tag bits, its size is minimal, but it needs an associativity proportional to the number of cores. Sparse directories limit the associativity of the directories by increasing the number of sets of the directory. However, because the distribution of addresses in a directory is not uniform, sparse directories increase the number of sets by a factor greater than the factor by which the associativity is reduced. Additionally, due to the reduced associativity of sparse directories, and consequently, the lack of correspondence between a directory entry and a block frame, entries in a sparse directory must also store a representation of sharers (usually a bit-vector of sharers) in addition to the tag bits.

Increases in the on-chip core count and associativity of core-private caches [16] have substantially increased the dynamic power consumption of the on-chip coherence directories. For every read miss in a private cache, a directory access must be performed in parallel with the shared last-level cache lookup to ensure that another private cache does not hold a more recent version of the accessed block. Similarly, for every write miss in a private cache, a directory access is needed to find any existing sharers of the written block to invalidate their copies. The growing core count increases the aggregate number of directory accesses, while both growth of the core count and the private cache associativity [16] increase the energy used by each directory access, with the two effects substantially increasing the power consumption of the aggregate on-chip directory as CMPs continue to scale.

3. WHY FILTER DIRECTORY LOOKUPS?

A directory handles coherence events specific to the coherence protocol. We examine the cases of read, write, upgrade, and eviction requests. For every private-cache miss, a read, write, or upgrade request is sent to the directory. For every evicted block from a private cache, an eviction request is sent to the directory. For upgrade and eviction requests from the private caches, a corresponding entry is known to exist in the directory, making a search operation unnecessary (i.e., a backward pointer in the cache can specify the location of the entry in the directory). Consequently, all directory lookups happen as a result of read and write requests. On a read request, a sharer must be added to an existing entry; on a write request, shared copies referenced in an existing entry must be invalidated; and, in both cases, a new entry must be allocated if a reference to the accessed block is not found during the lookup.

Because private caches are small, workload working sets are large, and multi-threaded workloads are highly optimized, most of the private-cache misses are capacity or compulsory misses that result in directory lookups that fail to find a corresponding entry in the directory [11]. Therefore, the majority of the lookups performed on read and write requests waste energy. We examined the directory access patterns of a 16-core system with a shared L2 cache and statically-interleaved 16-bank duplicate-tag directory (system and workload details can be found in Table 1). Figure 2 shows the directory accesses that did not find a corresponding entry in the duplicate-tag directory as a fraction of all read and write requests observed by the directory. We confirm that, on average, more than 69% of requests do not find the accessed block in the directory, wasting the majority of the energy spent on directory lookups. It should be noted that sparse directories waste a bigger fraction of

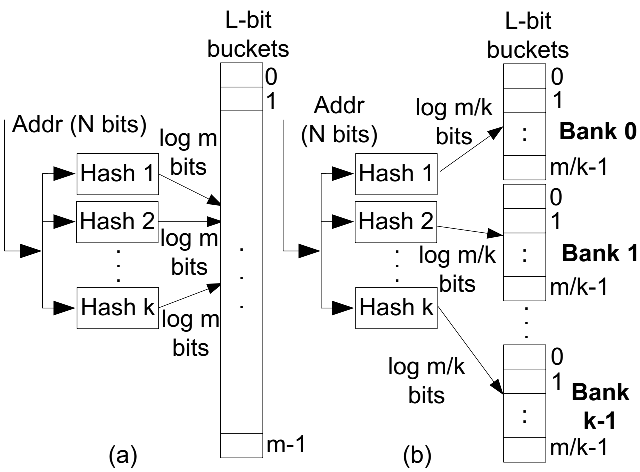

FIGURE 3. Bloom Filter. (a) Counting Bloom filter, (b) Parallel Bloom filter.

their power on useless lookups because, compared to duplicate-tag directories, they map fewer (at best map as many) entries.

Due to the fact that the vast majority of power-hungry directory lookups are not necessary, a mechanism to accurately determine if an entry is in the directory and consequently skip the needless directory lookups can significantly reduce power consumption.

4. TURBOTAG

We propose to split the coherence directory into two structures: a filter, responsible for determining whether or not the directory access will find a match and the directory that contains sharer information. The filter acts as power-efficient lookup structure, confirming that a match in the directory does not exist and eliminating the need for a costly directory access. As the basis of the filter, we use the Bloom filter, a space- and power-efficient data structure to test set membership.

4.1 Bloom Filters - Background

Bloom filters (or counting Bloom filters) are probabilistic data structures used to test whether an element is *not* a member of a set [3]. The structure of a counting Bloom filter is shown in Figure 3(a). The filter structure comprises an array of m buckets, initially all 0, and k hash functions (each bucket has L bits). To insert an element into the filter, k buckets corresponding to the element's hash values are incremented (each hash function produces a $log(m)$-bit index into the bucket array). A membership test checks the k buckets corresponding to the hashed values of the lookup key. If at least one of the k buckets is zero, the element is not in the set. If all k buckets are non-zero, membership is unknown and the element may be in the set. In case of an overflow in a bucket, the bucket's counter should not be decremented until all elements are removed from the counting Bloom filter.

A hardware implementation of a counting Bloom filter requires using an SRAM with k read/write ports or performing a multi-cycle operation to read and write the contents once per cycle for k cycles. A hardware realization of the Bloom filter technique that uses multiple independent parallel single-ported SRAMs [15] offers a less complex approach. The parallel Bloom filter implementation splits the bucket list into k independent banks (see Figure 3(b)). Each hash function is used with only one of the SRAM banks, independent of other hash functions. This approach

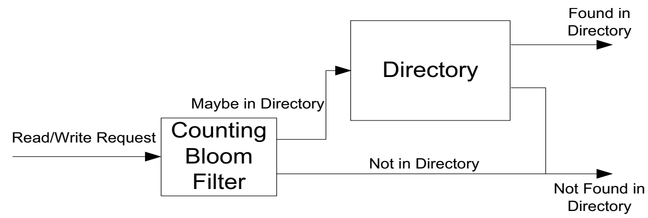

FIGURE 4. TurboTag overview.

has the advantage of using single-ported SRAMs and completing the insertion and removal operations without complex state machines. Although the implementation may result in a slight increase in the rate of false positive results from the filter, in practice, the probability of false positives is unaffected as long as k/m is much less than *1.0* [15].

4.2 TurboTag Design

The TurboTag augments every directory slice in a CMP with a counting Bloom filter to keep track of addresses in the underlying coherence directory as shown in Figure 4. Coherence events update the filter. When the last sharer evicts a block (via clean eviction or dirty writeback), the corresponding buckets in the filter are decremented. Upgrade requests do not consult the filter and are handled directly by the directory. Whenever a new entry is allocated in a directory as a result of a read or write event, the corresponding counters in the filter are incremented.

TurboTag enables a power-efficient lookup mechanism to assert that a lookup in the underlying directory is not needed. In case the filter indicates that an entry potentially exists in the directory, a directory search is performed to ensure correct operation of the system. On every read and write event, the filter is consulted prior to a search in the directory structure. If the filter indicates that the accessed block is not present in the directory, the directory lookup is skipped, reducing directory power consumption. The majority of read and write accesses to the directory search for blocks that are not present in the directory. Therefore, TurboTag eliminates the majority of unnecessary directory searches on these requests.

It is important to note that, the number of buckets in the filter and the number of elements added to it determine the rate of false positives. When the filter is undersized, it will more frequently indicate that an entry may be present in the set, when it is actually absent, degrading the effectiveness of the filter. However, regardless of filter ineffectiveness, correct system operation is always preserved.

5 EVALUATION

5.1 Methodology

We analyze the directory access patterns using simulation of a tiled CMP that executes unmodified applications and operating system in *FLEXUS* [19]. FLEXUS extends the *Virtutech Simics* functional simulator with models of processing tiles, NUCA cache, on-chip protocol controllers and on-chip interconnect. We summarize our tiled architecture parameters in Table 1 (left).

The simulated system runs the *Solaris 8* operating system and executes the workloads listed in Table 1 (right). We include a range of server and scientific workloads in our evaluation. We note that our system configuration is similar to modern hardware, although we conservatively select parameters that give TurboTag

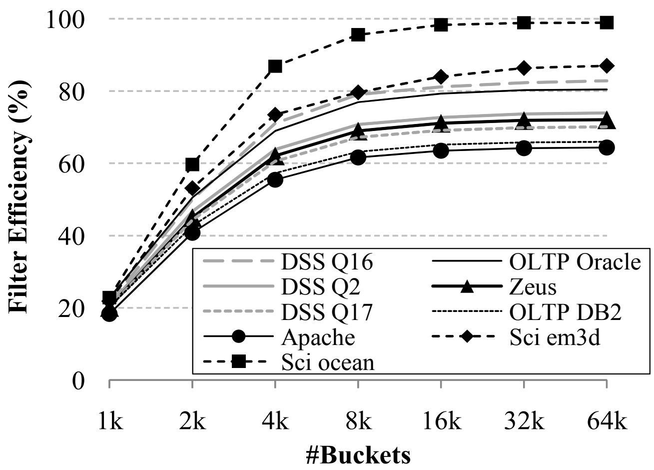

FIGURE 5. (left) Filter efficiency. The accuracy of the filter is shown as a function of Bloom filter size. The 64K filter is effectively ideal, capable of filtering all possible accesses.

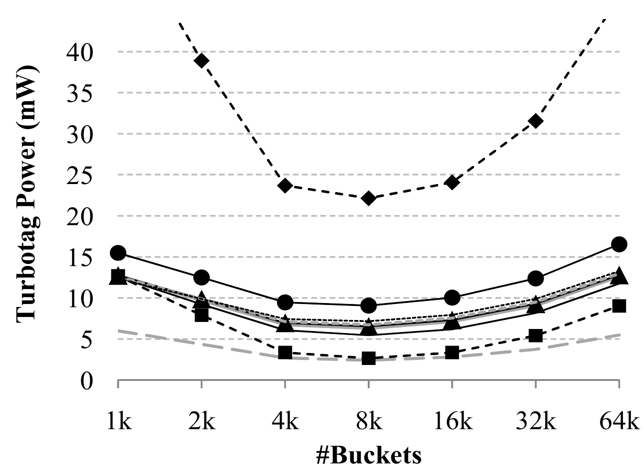

FIGURE 5. (right) TurboTag power dissipation sensitivity. Total directory power (TurboTag filter and coherence directory) is plotted as a function of the filter size.

the least benefit. For example, today's server processors use 2-, 4-, and even 8-way [16] set-associative L1 data caches. While private L1 caches with higher associativity allow more benefit from filtering directory access, we evaluate 2-way caches to stress the benefits of our approach even under pessimistic assumptions.

TurboTag benefits apply to today's systems using the duplicate-tag organization [2][13] and are even more effective for sparse directories [7] that are likely to be used in near-future designs. In this work, we limit our evaluation of TurboTag to the duplicate-tag directory only, noting that the benefits of TurboTag are strictly greater for sparse, limited [7], and other [21] directory organizations that do not store precise sharer information (see Section 2).

We use CACTI 5.3 [18] for all power estimates, using the 45nm technology node. Because directory accesses are not on the critical path of the system (see detailed analysis in Section 5.4), we assume ITRS-LSTP (low standby power) transistors to estimate the delay and power consumption. ITRS-LSTP transistors dissipate less static power compared to other models but are slower than their ITRS-HP (high performance) counterparts. To estimate directory power, we consider only the dynamic-power consumption of SRAM storage that contains directory tags, neglecting the power consumption of comparators and supporting logic. Our Bloom filter has two banks and uses 4-bit buckets. However, CACTI is limited to structures with a minimum output of 8 bits. Due to this limitation, we estimate the power of the Bloom filter by modeling an 8-bit SRAM structure and linearly scaling it to 4 bits.

5.2 Sensitivity Analysis

TurboTag performance is inversely proportional to the false-positives rate, depending primarily on the number of buckets used in the Bloom filter. Figure 5 (left) shows the Bloom filter effectiveness for a varying number of filter buckets for each workload. Beyond 32K buckets, the false-positives rate drops to zero, effectively reaching the maximum filter opportunity (Figure 2) at 64K buckets. Across all workloads, a nearly identical trend is visible in the relationship of the filter accuracy and the filter size. Significant accuracy gains can be seen as the filter size is increased to 8K buckets, with only marginal gains up to 16K and beyond.

To better demonstrate the power trade-offs of the TurboTag design, the filter efficiency data shown in Figure 5 (left) are used to compute the power consumption of TurboTag and the underlying directory and are shown together in Figure 5 (right). Although a greater number of buckets decreases the rate of false positives, which reduces the number of accesses to the underlying coherence directory and saves energy, accessing the larger filter structure

TABLE 1. System and application parameters.

CMP Size	16-cores
Processing Cores	UltraSPARC III ISA; 2GHz
L1 Caches	split I/D, 64KB 2-way 64-byte blocks, write-back
L2 NUCA Cache	1MB per core, 16-way 64-byte blocks
Main Memory	3 GB memory, 8KB pages
Memory Controller	one per 4 cores round-robin page interleaving
Interconnect	2D folded torus (4x4)

OLTP – Online Transaction Processing (TPC-C v3.0)	
DB2	*IBM DB2 v8 ESE,* 100 warehouses (10 GB), 64 clients, 2 GB buffer pool
Oracle	*Oracle 10g Enterprise Database Server* 100 warehouses (10 GB), 16 clients, 1.4 GB SGA
Web Server (SPECweb99)	
Apache	*Apache HTTP Server v2.0.* 16K connections, fastCGI, worker threading model
Zeus	16K connections, fastCGI
DSS – Decision Support Systems (TPC-H)	
Qry 2, 16, 17	*IBM DB2 v8 ESE*, 480 MB buffer pool, 1GB
Scientific	
em3d	768K nodes, degree 2, span 5, 15% remote
ocean	1026x1026 grid, 9600s relaxations, 20K res., err tol 1e-07

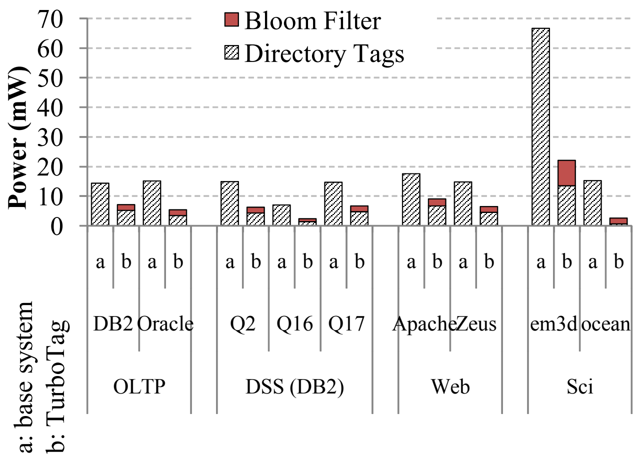

FIGURE 6. TurboTag power dissipation benefits. Total power (TurboTag filter and coherence directory) are presented along-side the base directory power consumption.

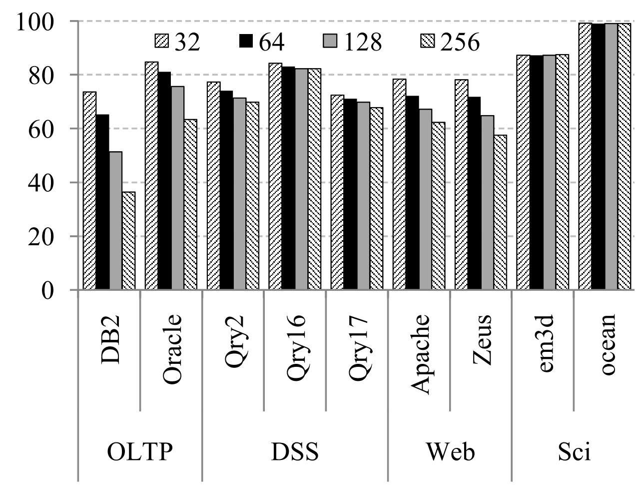

FIGURE 7. Filter applicability. Directory filtering opportunity for various private-cache sizes.

itself requires more energy. As increasing the number of buckets enables higher filtering accuracy, the combined power consumption of TurboTag and the underlying directory quickly decrease, leveling off at 8K. Beyond 8K buckets, the additional power dissipation of the filter is sufficient to overcome the benefit of marginally greater filter efficiency. We note that our directory-power model accounts only for the storage power dissipation and not for the power dissipation of comparators and other directory logic, which may further increase the per-access savings achieved by TurboTag, causing a larger filter design to yield maximal power savings. Additionally, we note that a 3-bit bucket design has low overflow rates; depending on the mechanism and frequency of clearing overflows, a 3-bit design may be beneficial overall, and would also push the optimal design point toward a larger filter size.

5.3 Power Savings

We compare the power dissipation of the baseline directory enhanced with TurboTag with 8K buckets and the power dissipation of the baseline directory without TurboTag. The dynamic power consumption is computed based on access frequency of each structure. For TurboTag, every directory access incurs four accesses to the Bloom filter, two to read filter results, and two to modify filter contents. Figure 6 shows a side-by-side comparison of the directory with and without TurboTag, further breaking down the power dissipation of TurboTag into the Bloom filter and directory components. We observe that the Bloom filter efficiently handles a large fraction of the directory requests, preventing them from accessing the underlying directory. On average, TurboTag reduces the directory power consumption by 58%.

5.4 Performance Considerations

Accessing a Bloom filter before the directory access affects the total directory latency. Although the accesses filtered by TurboTag are resolved more quickly than the base system's directory, accesses that are first checked by the Bloom filter and then by the underlying directory experience an increased latency. For accesses that consult the underlying directory, an extra delay of the Bloom filter is added, and for accesses that are filtered by the Bloom filter, the latency is reduced to the access time of the filter alone. We use CACTI to estimate the total access time of the Bloom filter and directory storage structures and compare it to the access time of the L2 cache slice which is performed in parallel with the directory lookup. For the worst case, the combined delay of the Bloom filter and the directory is approximately 2.3ns, whereas the tag access of a 1MB L2 cache slice requires 4.5ns. We conclude that, despite the additional filter latency, the directory access remains off of the critical path and does not negatively affect system performance.

5.5 Future Outlook

We estimate the impact of TurboTag on the directory power dissipation of future CMPs by examining the filtering opportunity for a range of private cache sizes. Figure 7 presents the per-workload filtering opportunity for 32KB and 64KB caches that resemble L1 caches built today; as expected, smaller cache sizes increase the filter opportunity as a result of a reduction in the aggregate storage capacity of the private caches. Additionally, we present filtering opportunity for 128KB and 256KB caches that are likely candidates for the lower level of the multi-level private hierarchies of today's CMPs [13][16]. We note that filtering opportunity remains high as the aggregate cache size is increased exponentially.

Overall, we find that multi-threaded server and scientific workloads tend to avoid sharing, leaving TurboTag filtering opportunity largely unaffected as the aggregate cache size increases. Unlike other workloads, OLTP DB2 exhibits a non-trivial amount of active sharing between threads, decreasing the filtering opportunity as the aggregate cache capacity increases. However, we expect that, as the number of on-chip cores grows, the latency of on-chip communication will push developers to further optimize code and avoid thread communication, a change that will further increase TurboTag filtering opportunity.

6. RELATED WORK

In this work, we examined the potential of TurboTag to reduce the power consumption of the duplicate-tag coherence directory from Barroso et al. [2]. However, the TurboTag technique directly applies to other directory organizations such as the sparse directory organization [7] from Gupta et al. and the tagless directory [21] from Zebchuk et al. Kin et al. used a small filter cache [9] to reduce power consumption of a larger cache. Whereas filter caching relies on locality of reference of filter requests, TurboTag filters requests that have no locality in the directory accesses, filtering requests that will not be found in the directory. The behavior of TurboTag is therefore more closely resembles detecting virtual synonyms with a Bloom filter [20] proposed by Woo et al. and predicting cache misses [12] explored by Peir et al.

A number of prior mechanisms relied on the tendency of coherence requests to not find a matching entry. Moshovos et al. proposed Jetty [11] to filter snoop requests and RegionScout [10] that relied on regions of addresses having similar behavior. Jetty was later evaluated by Ekman et al. in the context of CMPs [6]. Many other researchers evaluated reducing snoop traffic via filtering. Cantin et al. proposed region-based mechanisms to avoid coherence traffic [4]. Salapura et al. proposed improving snoop filter accuracy [14] by taking advantage of temporal locality in snoop requests. Strauss et al. implemented snoop filtering [17] for improved performance and energy. Chinthamani and Iyer evaluated snoop filters on small-scale SMPs [5]. Ballapuram et al. used Bloom filters and software hints [1] to reduce snoop traffic. Rather than filtering snoop messages or snoop-induced lookups in small-scale systems, TurboTag reduces coherence activity energy in the coherence directory of larger-scale CMPs.

7. CONCLUSIONS

In this work, we showed that coherence directories dissipate a significant fraction of their power on needless coherence checks. Although coherence checks on each L1 cache miss are required for correct operation, 69% of the private-cache misses find no sharer in the directory, effectively wasting the energy for the coherence checks. To reduce wasted energy in the directory, we proposed TurboTag, a counting Bloom filter to eliminates needless directory accesses. We evaluated the design of TurboTag on full-system traces of server and scientific workloads and found that TurboTag avoids nearly all directory searches for non-shared blocks, eliminating over 69% of accesses to the underlying directory and achieving a 58% reduction in the dynamic power consumption.

REFERENCES

[1] C.S. Ballapuram, A. Sharif, and H.S. Lee, “Exploiting Access Semantics and Program Behavior to Reduce Snoop Power in Chip Multiprocessors,” ASPLOS XIII: Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems, New York, NY, USA: 2008.

[2] L.A. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, and B. Verghese, “Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing,” ISCA '00: Proceedings of the 27th International Symposium on Computer Architecture, New York, NY, USA: 2000.

[3] B.H. Bloom, “Space/Time Trade-offs in Hash Coding with Allowable Errors,” Communications of the ACM, vol. 13, 1970, pp. 422-426.

[4] J.F. Cantin, M.H. Lipasti, and J.E. Smith, “Improving Multiprocessor Performance with Coarse-Grain Coherence Tracking,” ISCA '05: Proceedings of the 32nd annual international symposium on Computer Architecture, Washington, DC, USA: 2005.

[5] S. Chinthamani and R. Iyer, “Design and Evaluation of Snoop Filters for Web Servers,” Proceedings of the 2004 Symposium on Performance Evaluation of Computer Telecommunication Systems, San Jose, CA, USA: 2004.

[6] M. Ekman, F. Dahlgren, and P. Stenström, “Evaluation of Snoop-Energy Reduction Techniques for Chip-Multiprocessors,” Proceedings of the First Workshop on Duplicating, Deconstructing, and Debunking, Anchorage, Alaska: 2002.

[7] A. Gupta, W. Weber, and T. Mowry, “Reducing Memory and Traffic Requirements for Scalable Directory-Based Cache Coherence Schemes,” ICPP '90: Proceedings of the 1990 International Conference on Parallel Processing, Urbana-Champaign, IL, USA: 1990.

[8] N. Hardavellas, I. Pandis, R. Johnson, N.G. Mancheril, A. Ailamaki, and B. Falsafi, “Database Servers on Chip Multiprocessors: Limitations and Opportunities,” Conference on Innovative Data Systems Research, CA, USA: 2007.

[9] J. Kin, M. Gupta, and W.H. Mangione-Smith, “The Filter Cache: An Energy Efficient Memory Structure,” MICRO 30: Proceedings of the 30th ACM/IEEE international symposium on Microarchitecture, Washington, DC, USA: 1997.

[10] A. Moshovos, “RegionScout: Exploiting Coarse Grain Sharing in Snoop-Based Coherence,” CA '05: Proceedings of the 32nd annual international symposium on Computer Architecture, Washington, DC, USA: 2005.

[11] A. Moshovos, G. Memik, B. Falsafi, and A. Choudhary, “JETTY: Filtering Snoops for Reduced Energy Consumption in SMP Servers,” HPCA '01: Proceedings of the 7th International Symposium on High-Performance Computer Architecture, Washington, DC, USA: 2001.

[12] J. Peir, S. Lai, S. Lu, J. Stark, and K. Lai, “Bloom Filtering Cache Misses for Accurate Data Speculation and Prefetching,” ICSC '02: Proceedings of the 16th international conference on Supercomputing, New York, NY, USA: 2002.

[13] S. Patel, S. Phillips, and A. Strong, “Sun's Next-Generation Multithreaded Processor - Rainbow Falls,” Hot Chips 21, Stanford, CA, USA: 2009.

[14] V. Salapura, M. Blumrich, A. Gara, I.B. Thomas, and J.W. Research, “Improving the Accuracy of Snoop Filtering Using Stream Registers,” MEDEA '07: Proceedings of the workshop on MEmory performance, New York, NY, USA: 2007.

[15] D. Sanchez, L. Yen, M.D. Hill, and K. Sankaralingam, “Implementing Signatures for Transactional Memory,” MICRO '07: Proceedings of the 40th IEEE/ACM International Symposium on Microarchitecture, Washington, DC, USA: 2007.

[16] R. Singhal, “Inside Intel® Next Generation Nehalem Microarchitecture,” Hot Chips 20, Stanford, CA, USA: 2008.

[17] K. Strauss, X. Shen, and J. Torrellas, “Flexible Snooping: Adaptive Forwarding and Filtering of Snoops in Embedded-Ring Multiprocessors,” ISCA '06: Proceedings of the 33rd international symposium on Computer Architecture, 2006.

[18] S. Thoziyoor, N. Muralimanohar, J.H. Ahn, and N.P. Jouppi, “CACTI 5.1,” 2008.

[19] T.F. Wenisch, R.E. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, and J.C. Hoe, “SimFlex: Statistical Sampling of Computer System Simulation,” IEEE Micro, vol. 26, 2006, pp. 18-31

[20] D.H. Woo, M. Ghosh, E. Ozer, S. Biles, and H.S. Lee, “Reducing Energy of Virtual Cache Synonym Lookup using Bloom Filters,” CASES '06: Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems, New York, NY, USA: 2006.

[21] J. Zebchuk, V. Srinivasan, M.K. Qureshi, and A. Moshovos, “A Tagless Coherence Directory,” MICRO '09: Proceedings of the 42st IEEE/ACM International Symposium on Microarchitecture, New York, NY, USA: 2009.