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.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ISLPED’10, August 18–20, 2010, Austin, Texas, USA.
Copyright 2010 ACM 978-1-4503-0146-6/10/08...$10.00.
private caches [16] have substantially increased the dynamic in parallel with the shared last-level cache lookup to ensure that read miss in a private cache, a directory access must be performed power consumption of the on-chip coherence directories. For every another private cache does not hold a more recent version of the increases in the on-chip core count and associativity of core-sharers (usually a bit-vector of sharers) in addition to the tag bits. entries in a sparse directory must also store a representation of correspondence between a directory entry and a block frame, which the associativity is reduced. Additionally, due to the reduced increase the number of sets by a factor greater than the factor by private cache associativity). Because a duplicate-tag directory as a fraction of all read and write requests accesses 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 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.
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 shared 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 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

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
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. |
|------------------|----------------|
| **OLTP – Online Transaction Processing (TPC-C v3.0)** | **DB2** |
| | IBM DB2 v8 ESE, 100 warehouses (10 GB), 64 clients, 2 GB buffer pool |
| Oracle | 100 warehouses (10 GB), 16 clients, 1.4 GB SGA |
| Web Server (SPECweb99) | **Apache HTTP Server v2.0.** |
| | Apache 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 |
power consumption is computed based on access frequency of the baseline directory without TurboTag. The dynamic directory storage structures and compare it to the access time of the Bloom filter and 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

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

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].

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

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.
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.


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 eliminate 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