The Effects of Granularity and Adaptivity on Private/Shared Classification for Coherence

Mahdad Davari  
Uppsala University, Sweden  
mahdad.davari@it.uu.se

Alberto Ros  
University of Murcia, Spain  
ares@ditec.um.es

Erik Hagersten  
Uppsala University, Sweden  
erik.hagersten@it.uu.se

Stefanos Kaxiras  
Uppsala University, Sweden  
stefanos.kaxiras@it.uu.se

Abstract

Classification of data into private and shared has been used by coherence protocols based on properties of relaxed memory consistency and data-race-free semantics, where each core downgrades and invalidates its shared data at synchronization points – locks/barriers. Less shared data is therefore expected to improve the coherence performance. In this work we investigate the impact of data classification granularity – page vs. block – and adaptation – shared to private – on coherence performance, since these metrics result in less data classified as shared. Our findings reveal that although fine-grained and adaptive data classification result in significantly more private data, the impact on coherence performance is insignificant. This is due to the fact that the increased private data is not re-accessed before being evicted from the cache, hindering performance gains of having more private data. The network traffic, however, shows more sensitivity, with reduction up to 30% in a number of benchmarks.

1. Introduction and motivation

Many approaches use private/shared data classification to simplify coherence [5, 15]. Private data can be excluded from the burden of coherence, resulting in directory size reduction [14, 8, 3] or simplifying/optimizing the coherence protocol itself [15]. While some approaches rely on page-based private/shared information provided by the operating system (OS) [15], others maintain multigranular classification mechanism in hardware [3, 18]. Page-level classification suffers from mis-classifying the private cache-lines as shared in shared pages. Furthermore, without adaptive classification, which allows the shared data to be re-classified as private, all the data in the system are eventually misclassified as shared, eliminating optimizations connected with the private data. The increase in the amount of shared data directly translates into performance loss and higher energy consumption, as shared data triggers the coherence mechanism, enforcing coherence invalidations, delays and traffic. In this work we study how the performance of coherence protocols based on private/shared data classification is affected by the classification granularity and adaptation.

To perform the study, we design a robust hardware-based data classification approach called generational classification. Our classification is based on the precise definition of generational behavior of cache lines [17]. A generation begins when a cache line is brought into a core’s first-level cache (L1) due to a cache miss. A cache line can have multiple concurrent generations up to the number of cores in the system. A cache line is classified as private if there exists only one generation of that cache line across all the L1 caches. We further combine generational data classification with self-invalidation and self-downgrade [15, 5] to develop a coherence protocol called generational coherence (GC), which we use to investigate the benefits of fine-grained and adaptive data classification on the chosen family of the coherence protocols. While existing software-based approaches suffer from coarse classification granularity as well as being OS-dependent [15], our approach performs the classification independent of OS at the inherent cache-line granularity. Furthermore, our approach provides classification adaptation without incurring complexities and overhead of multigranular mechanisms [3, 18].

2. Background and Related Work

2.1. VIPS-M

Among recent proposals advocating simple coherence for data-race-free semantics [5, 15] we choose the VIPS-M coherence protocol [15] as our baseline. VIPS-M is a directory-less protocol that allows incoherence in-between synchronizations, but sequential consistency for data-race-free programs [1]. VIPS-M classifies data at page granularity using the OS and TLBs [6, 10]. A page starts as private in the page table, and becomes permanently shared upon an access from a second core. Cache lines belonging to private pages are not affected by synchronizations, whereas each core invalidates its L1 shared cache-lines after acquiring locks or crossing barriers. This self-invalidation is performed in bulk by setting the invalid bit of all the shared cache lines. Modified shared cache-lines use write-through policy to update the shared last-level cache (LLC) with the modified data – self-downgrade. Each core should guarantee that self-downgrade of all the modified shared data is complete – i.e. modified data are visible in the shared LLC – before a core can release a lock or cross a barrier [15].

2.2. Hardware private/shared classification approaches

Many systems rely on a hardware private/shared classification for a variety of reasons. Chief among these reasons is the reduction of the directory size [3, 18, 7]. The underlying approach is to implement a multi-grain directory that keeps track of coherence information on more than one block size. Alisafaei [3] and Zebchuk et al. [18] propose similar approaches that differ in implementations. The idea is to store
private regions in the directory and out of those regions extract cache lines that are shared. Similarly, Fang et al. [7] describe an approach where region entries can be private but block entries extracted from these regions can either be private — with a different owner than the region — or shared. All the aforementioned approaches incur complexity by suggesting multi-granular methods.

Regarding adaptivity of classification Pugsley et al. introduce an adaptive version of their SWEL protocol, called Reconstituted SWEL (RSWEL) that includes a 2-bit saturating decay counter to re-classify shared pages to private [14]. However, the adaptation is complex, requires elaborate tuning of the time period that ticks the decay counters for good results, and lastly it is only initiated periodically in bulk.

3. Generational Classification

Data classification is performed in the LLC since LLC observes all the requests from L1 caches. Classification is stored in the LLC tags using a Private/Shared bit and a dual purpose PrivateOwner/SharerCount field which holds the Owner ID for private data, or the number of sharers for shared data. Classification is carried along with data response from LLC to the cores. After a generation for a cache line is started in an L1 cache, the cache line may be repeatedly accessed by the core before entering a "dead" time awaiting eviction. A generation terminates when the cache line is evicted from L1. We use replacements as approximation for termination of generations.

Upon receiving the first request for a cache line in the LLC, the cache line is classified as <Private, Owner>. Upon receiving the second request for the same cache line from a new core, the former owner is notified by the LLC to change the internal classification to shared, and consequently the cache line is re-classified as shared <Shared, 0> in the LLC. The process of private-to-shared transition requires more consideration, which is addressed in the next paragraph. Further accesses increment the degree of sharing <Shared, n+1>. To be able to adapt the classification back to private, evictions from L1 should be made visible to the LLC in order to be able to decrement the degree of sharing <Shared, n>. Replacement of modified private and shared blocks from the L1s is detected by observing write-backs in the LLC. Clean shared blocks notify the LLC of their eviction from L1 via explicit eviction notifications (EEN). Classification adapts back to private in the LLC when degree of sharing reaches zero. The first core requesting a cache line classified as <Shared, 0> receives the cache line in private state.

We allow silent replacement of clean private cache lines. It is more efficient not to track the end of the generation for those blocks since private data, which do not require adaptation, constitute the majority of data in the system. Upon receiving a request in the LLC from a new core for a cache line already classified as private we detect through a mechanism called recovery whether the former owner still owns the block — cache line becomes shared, – or the private cache line has been clean and silently replaced by the owner – cache line remains private with a new owner. The former owner is notified by the LLC. If the owner still has the line, it re-classifies the line to shared and performs a write-back of dirty data or sends an acknowledgement if line is clean. However, owner replies with a negative acknowledgement if line is silently evicted, which leaves the cache line as private with a new owner.

4. Generational Coherence

GC borrows self-invalidation/self-downgrade from VIPS-M and combines it with generational data classification. We allow silent self-invalidations in L1s to eliminate the EEN traffic for self-invalidations. A self-invalidated cache line is not evicted from L1. Eviction of a self-invalidated cache line due to replacement informs the LLC via an EEN. Re-accessing a self-invalidated cache line should not create a new generation, since a generation already exists in the LLC. Therefore, re-accessing a present self-invalidated cache line should be distinguished from accessing a cache line that is not present in L1. In the latter case L1 issues to LLC a regular GET request which starts a new generation, whereas in the former case a REFRESH request is issued, which leaves the classification unaffected.

5. Evaluation Methodology

We evaluate GC against a directory protocol (MESI states) and VIPS-M. We use the Simics [12] and GEMS [13] simulators. We also employ GARNET [2] to model the interconnection network. Our target system is a 16-tile chip multiprocessor. We use Pin [11] in order to study and analyze the shared data accesses. We employ a variety of parallel applications belonging to SPLASH-2 [16] and PARSEC [4] benchmark suites. We also implement a version of the protocol which uses cache decay as a simple dead-block predictor [9] to investigate its impact on adaptation. Cache decay can potentially result in early detection of dead blocks during private-to-shared recovery process. However, our findings reveal that it does not have significant impact for our benchmarks.

6. Results

6.1. Classification Quality

6.1.1. Insensitive Applications to Granularity and Adaptivity

Fig.1 shows two applications that do well with page-level classification. The Y-axis shows the percentage of accesses that are made to the shared data up to each point of program execution. Lower values on Y-axis translate into less accesses to shared data, hence less total shared data in the system. As the figure shows, granularity or adaptivity have no significant impact on the outcome of classification for those benchmarks. This is the case where most of the cache lines in a page classified as shared are truly shared.

6.1.2. Applications Sensitive to Granularity

Fig.2 shows two benchmarks that benefit from fine-grained classification.
Fine-grained classification prevents private cache lines from being misclassified as shared in shared pages. The benchmark in Fig.2-b also slightly benefits from adaptation.

6.1.3. Applications Sensitive to Granularity and Adaptivity. Benchmarks in Fig.3 benefit from both fine-grained classification and adaptation. Adaptation serves to prevent temporarily private data from being misclassified as shared [3].

Despite no significant change in miss rate and performance, Fig.8 shows that GC reduces the network traffic up to 30% in Watersp, and about 20% in FFT, FMM, and Swaptions benchmarks. Fig.9 explains this by showing the reduction in the amount of write-through traffic for two of the benchmarks. Less shared data translates into less self-downgrade in form of write-through. There are also benchmarks, such as LU-NCB, Ocean-CP and Blackscholes, where the increase in write-back traffic due to frequent replacements cancels out the benefit of having less write-through traffic. Benchmarks such as Barnes, Raytrace, Volvernd, and Water-nsq in Fig.8 are less sensitive to GC and experience more traffic. The increased traffic comes from the EEN and recovery messages without resulting any reduction in write-through traffic. Fig.6 and Fig.8 together confirm that GC’s network traffic reduction is more sensitive to write-through rather than LLC-to-L1 data movement, as GC’s impact on miss rate is insignificant.

7. Conclusions

- for many benchmarks adaptive block-level classification significantly reduces the amount of shared data, yet does not affect the overall miss rate (Fig.6).
- the data re-classified as private using adaptive block-level classification, which would have otherwise been classified as shared at page granularity, in many cases, cause reduction of write-throughs and network traffic (Fig.8).

How can these two seemingly contradictory observations be reconciled? The answer lies in the dynamics of the generational behavior. Most of the data re-classified as private
dead before synchronization and are not re-accessed immediately after. In other words, the live time of a single generation of such data does not typically span across synchronization points. This is shown by the small re-access rate of self invalidated data (Fig.7) which means that the behavior of the miss rate is dominated by capacity/conflict misses. Any change in self-invalidation misses is hardly noticeable —except when magnified by very frequent synchronization as in the case of Radiosity. However, while most of the data re-classified by generational coherence as private are dead at synchronization points, this does not mean that they are not re-accessed at a much later time, starting a new generation. The compound effect of such private generations is to reduce write-through traffic. When this effect is not balanced out by write-back traffic or control overhead, the overall network traffic is reduced.

References