Fine-Grain Data Classification to Filter Token Coherence Traffic

Bhargavi R. Upadhyay[a], Alberto Ros[b], Supriya M.[a]

[a]*Department of Computer Science and Engineering, Amrita School of Engineering, Bengaluru, Amrita Vishwa Vidyapeetham, INDIA*
[b]*Computer Engineering Department, University of Murcia, Murcia, SPAIN*

Abstract

Snoop-based cache coherence protocols perform well in small-scale systems by enabling low latency cache-to-cache data transfers in just two-hop coherence transactions. However, they are not a scalable alternative as they require frequent broadcast of coherence requests. Token coherence protocols were proposed to improve the scalability of snoop-based protocols by removing a large amount of traffic due to broadcast responses. Still, broadcasting coherence requests on every cache miss represents a scalability issue for medium and large-scale systems.

In this paper, we propose to reduce the number of broadcast operations in Token coherence protocols by performing an efficient fine-grain private-shared data classification and disabling broadcasts for misses to data classified as private. Our fine-grain classification is orchestrated and stored by the Translation Look-aside Buffers (TLBs), where entries are kept for a longer time than in local caches. We explore different classification granularity accounting for different storage overheads and their impact on filtering coherence traffic. We evaluate our proposals on a set of parallel benchmarks through full-system cycle-accurate simulation and show that a subpage-grain classification offers the best trade-off when accounting for storage, traffic, and performance. When running a 16-core

*Corresponding author
Email addresses: `bhargavi.upadhyay@gmail.com` (Bhargavi R. Upadhyay), `aros@ditec.um.es` (Alberto Ros), `m_supriya@blr.amrita.edu` (Supriya M.)

configuration, our subpage-grain classification eliminates 40.1% of broadcast operations compared to not performing any classification and 13.7% of broadcast operations more than a page-grain data classification. This reduction translates into less network traffic (16.0%), and finally, performance improvements of 12.0% compared to not having a classification mechanism.

Keywords: multicore, cache coherence, private-shared data classification, token protocol

1. Introduction

Coherence protocols can be classified mainly into two categories [1]: snoop-based coherence protocols [2, 3, 4, 5] and directory-based coherence protocols [6, 7, 8]. There are many variants of each group and even hybrid protocols [9]. Snoop-based protocols use broadcast messages, and can transfer data from one cache to another in just two hops. Snoop-based cache coherence protocols are cost-effective and straightforward for small-scale systems. However, they do not achieve good performance with an increasing number of cores due to the frequent issue of broadcast operations. On the other hand, directory-based coherence protocols avoid broadcast by using a directory to track the sharers of data blocks. However, the directory structure incurs non-scalable storage in its simpler form, and these protocols perform cache-to-cache transfers in three hops.

With the premise cache coherence protocols should avoid both indirection when performing a cache-to-cache miss and interconnect ordering, Token coherence [10] was proposed. Token coherence makes snoop-based protocols more scalable by reducing the network traffic using tokens. Still, Token coherence protocols issue a broadcast request on each cache miss. Our work departs from the observation that private data (i.e., accessed by a single core) do not require broadcasting requests in a Token protocol as other cores do not cache a copy of the requested data [4]. However, knowing the private nature of the data before or in parallel to cache access is a current challenge.

There have been many studies that focus on the private-shared classification of data to improve the cache coherence protocols driven by the fact that private and read-only shared data do not require a cache coherence mechanism [4, 11, 12, 13, 14, 15, 16, 17, 18, 19]. There are many proposals in the recent literature on the topic of data classification that differ based on the level they are obtained: compiler [20, 12], operating system [11], and hardware [21, 22, 23, 8, 14]. Compiler alternatives are less accurate due to the lack of sharing information at compile time. Operating system and hardware alternatives that perform the classification before issuing a coherence request work at coarse granularity (e.g., page grain) [4, 11, 14, 24].

Proposal.

Our private/shared data classification is carried out by Translation Lookaside Buffers (TLBs) communication, mostly on TLB misses. When a block is classified as private, the broadcast request issued to maintain coherence can be filtered. In particular, we evaluate the following classification schemes and their impact on filtering Token coherence traffic:

- Page-Grain Classification (PGC): This classification mechanism classifies data with page granularity through TLB-to-TLB communication [14]. Thus, PGC is able to remove the broadcast operation for requests to blocks belonging to a private page. It stores the private-shared page information in the TLB using a single bit per entry. PGC has a low classification accuracy as it faces a false-sharing problem, where two blocks belonging to the same page but privately accessed by different cores are classified as shared.

- Block-Grain Classification (BGC): This classification also performs the classification through TLB-to-TLB communication but works at a finer granularity than PGC, classifying memory blocks as private or shared [15]. It stores the private/shared information of each block by allocating bit vectors in each TLB entry. BGC helps to remove the miss classification of the blocks encountered in PGC, thus achieving more accuracy and, in the end, filtering more broadcasts than PGC. However, it requires more storage as it stores 64 bits (since there are 64 blocks in a 4KB page) per vector.

- Sub-page Grain Classification (SGC): This classification works at the granularity of a group of memory blocks. Its goal is to reduce the size of the bit vectors used in BGC, hence its total storage requirements. We analyze the impact of clustering memory blocks for classification purposes on both classification accuracy and performance of the Token coherence compared to the previous granularities.

Results. Using full-system simulation and 14 parallel applications from SPLASH-2 and PARSEC benchmarks we evaluate how the three classification approaches can improve Token protocol performance and compare them to a baseline Token protocol without any classification scheme for 8 cores, 16 cores, and 32 cores. We show that PGC has poor classification accuracy, while BGC has high memory requirements. In contrast, our proposed sub-page granularity (SGC) obtains the best of both worlds: the low memory requirements of PGC and the accuracy of BGC. Considering 16 cores, SGC reduces the broadcast operation by 40.1% (13.7% more than PGC), reducing network traffic by 16.0% (6.0% with respect to PGC). This improves the overall execution time of a Token protocol without data classification by 12.0% for 16 cores and 25.3% for 32 cores.

Contributions. The main contributions of this paper are:

- Proposing an intermediate granularity (sub-page) to perform private/shared classification of data blocks, that improves accuracy over a state-of-the-art page-grain classification mechanism, without impacting significantly network traffic.

- Integrating three different private/shared data classification techniques with Token coherence, and showing that they can filter broadcast traffic considerably, thus reducing network traffic and improving performance.

The rest of the paper is organized as follows. Section 2 talks about the background on data classification and Token coherence. Section 3 presents the different classification schemes evaluated in this work. Section 4 details how to use the classification mechanisms in Token protocols to filter coherence traffic. Section 5 presents the simulation environment, and Section 6 analyzes experimental results. Section 7 describes the most relevant related work, and finally, Section 8 concludes the paper.

2. Background

This section explains the basics of Token coherence and offers a summary of prior work on data classification.

2.1. Token coherence

Martin et al. [10] proposed TokenB, a Token-based coherence protocol that brings together advantages from snoop-based and directory-based protocols, that is, cache-to-cache low latency misses and the ability to work on any network topology. Cache coherence in Token coherence is maintained by a number of tokens (commonly as many tokens as cores in the system) assigned to every memory block. An owner token takes care of sending the data block when requested. Token protocols assure coherence protection by token counting: A core with all tokens gets exclusive access to the block, and a core with at least one token gets shared access to the block.

Cache misses generate coherence requests that are broadcast to all cores. In case of write miss, an exclusive request is generated, and all cores answer with all tokens they have. The core holding the owner token sends the data block. The requester processor gets exclusive permission for the block when it receives all tokens. In case of read misses, the core with the owner token needs to answer with the requested data, and other cores with more than one token could send tokens too. On evictions, Token protocols send all tokens assigned to a block along with the data block (if dirty) to the home node.

When several cores request exclusive permission simultaneously by issuing broadcast requests, it is probably that none of them obtain all tokens. If, after some attempts, this racy situation still occurs, cores will issue a Persistent request, which is serialized by an arbiter.

Unfortunately, the use of broadcasts on every cache miss increases network traffic and, therefore, power consumption in the interconnection network, contributing to overall power consumption.

2.2. Private-shared memory block classification

Data classification mechanisms are becoming popular for multi-core and heterogeneous computers as they permit sharing status-based many optimizations regarding block management, being tested already in commodity processors [25]. Private-shared data classification can be done at different levels: compiler, hardware, and operating system level. Our technique to filter broadcast requests requires the data classification to be known before a coherence transaction is generated after the cache miss. While hardware classification schemes are among the most accurate due to the run-time knowledge and the fine-grain classification, they commonly provide the classification after the coherence request has been generated (e.g., at the directory [22, 26]). We focus, therefore, on the subset of hardware classification techniques that are able to provide an accurate classification before the coherence protocol is *called*: those based on TLB-to-TLB communication.

TLB-based classification schemes detect data classification at run time derived from the classification information cached at the TLBs [14, 27]. The TLB-based classification relies on querying the other TLBs in the system about their use of the data. The TLB-to-TLB communication with requests and responses uses the same interconnect as the cache coherence protocol. A broadcast message is sent to all the other TLBs in the system in every TLB miss. They reply with their page usage information and the page translation if they hold it, which quickens the page table walk process, since communication between cores is faster than accessing the page table [28, 29, 30]. The page table walk occurs parallel to the TLB broadcast, as the address translation is obtained from the page table if none of the TLBs holds the translation. A recovery process may be triggered depending on the optimization enabled by the classification when a page nature changes from private to shared (e.g., on the first access of a core to a memory page).

TLB-based classification can also be performed at block granularity by adding bit vectors to each TLB entry and tracking the classification in a per-block basis [15, 31]. Finer granularity helps to remove the classification accuracy and the cost of extra hardware overhead and has been shown helpful in improving the performance and scalability of directory-based protocols. This work analyzes the benefits of this classification, along with a new classification that clusters blocks, on Token coherence.

3. TLB-based data classification schemes

We propose to apply TLB-based fine-grain data classification to filter Token coherence traffic. For this purpose, we describe in detail three alternatives for classifying data using TLB-to-TLB communication. The first classification mechanism works at page granularity and was proposed by Ros et al. [14] to improve directory-based coherence protocols. The second classification mechanism works at block granularity and was recently proposed by the authors of this work [15] also to improve directory-based coherence protocols. Finally, the third mechanism is a new proposal, inspired by the work by Soltaniyeh et al. [32] but placed at the TLB and performed through TLB-to-TLB communication without requiring any operating system modification, that works at an intermediate granularity and seeks to achieve a good balance in classification accuracy versus storage requirements.

3.1. Page-grain classification (PGC)

The page-grain classification mechanism classifies pages as private or shared using a TLB classification protocol and stores the classification information along with the TLB entries. The classification is based on gathering information about the use of the pages by each core. A simple approach is to treat a page present in the TLB as in use. Pages used by more than two cores are considered as shared, while pages used by a single core are considered as private.

This technique faces a mis-classification of memory blocks. Two blocks belonging to the same page but accessed by different cores would be considered as shared, as they belong to a shared page. The next schemes aim to avoid this mis-classification of blocks, making the data classification more accurate.

3.1.1. Classification protocol

The goal of the classification protocol is to classify each page as private or shared. The classification is stored along with the TLB entries, and therefore, on a TLB miss the classification protocol is triggered to retrieve the private-shared information.

With the purpose of collecting the information about the use of the pages by other cores, on each TLB miss, a broadcast request is sent to all cores in the system. This request is simply asking about the use of the page by other cores. All cores reply with the information about the use of the page, by sending the address translation for the page in case they have it in their TLB, which helps to accelerate TLB misses as the page table walk to find the address translation may incur long latency. Once the requesting core receives all the replies, it can classify the page. If no other core is using the page, the page is private. Otherwise, the page is shared.

In case of a race condition where a TLB that initiated a broadcast request receives a remote request from another TLB for the same page, the page is classified as shared by both cores.

Protocol messages and their size. The TLB request message broadcast to other cores TLBs includes the address of the accessed page and the requesting core information. These fields are standard in all protocol messages and included in the message header (8 bytes). Other core TLBs respond with a control message

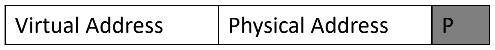


Figure 1: TLB entry for Page-Grain Classification

containing the header (8 bytes) plus the address translation (4 bytes) only if the requested page is present in the TLB.

3.1.2. Storage requirements

The classification information for each page is stored along with the TLB entries. Figure 1 shows how each TLB entry is extended to support page-grain classification (extra fields in gray). The virtual and physical addresses are already present in the TLB entries. The P bit (private) indicates if the page is private (P=1) or shared (P=0). The TLB is accessed on each cache access since both structures are accessed in parallel.

In case of a cache miss (and TLB hit), a coherence transaction is initiated for the missing block using the P bit corresponding to its page retrieved from the TLB before the cache miss is detected.

3.2. Block-grain classification (BGC)

Block-grain classification aims to improve the accuracy of page-grain schemes by tracking private-shared information for each accessed block. They are based on collecting information about the use (or predicted use) of the blocks by each core. TLB entries, therefore, need to be extended with information about the use of each block within the page: the *access* (A) bit vector. Each bit in the vector represents each of the blocks in the page. A bit set to one means that the block has been accessed, while a zero means that the block has not been accessed. The obtained per-block classification is also stored in another bit-vector field added to each TLB entry: the *private* (P) bit vector. The value of the corresponding bit for a memory block in both bit vectors indicate its classification status, as described in table 1.

Table 1: Access and private bits for a particular memory blocks and its classification status

Access	**Private**	**Classification status**
0	0	Block not accessed. No classification performed.
0	1	Block not accessed. But no other core asked for it, so it will become private when accessed.
1	0	Block accessed and shared.
1	1	Block accessed and private.

3.2.1. Classification protocol

The classification protocol is triggered in two situations: a TLB miss (since there are no vectors stored for that page) and TLB classification miss (when the corresponding bits to the accessed block in the access and private bit vectors are set to zero (0,0) and the classification is not know for that block). The protocol works as follows.

On a TLB or classification miss, a request is broadcast to all other TLBs in the system. All TLBs reply with their prediction of use for each block in a bit vector in case of a TLB miss, or simply the predicted use of the requested block in case of a classification miss. Classifying as many blocks as possible in a single protocol transaction helps to reduce the traffic overhead of the classification scheme. Similarly to the page-grain scheme, the address translation of the page is sent along with the responses to accelerate the TLB misses in case the address translation was not found in the local TLB.

Once the broadcasting TLB receives all replies, it extracts the information or predicted use of all other cores and performs a bit-wise NOR operation with the vectors received. The result from the operation indicates the privacy of each block in the page. That is, only if no other core claims a use of the block, it can be considered private.

Figure 2 shows an example of the classification protocol for the block-grain scheme, which also includes some optimizations explained along with the example. Core-3 suffers a TLB miss and, consequently, it broadcasts a request for a

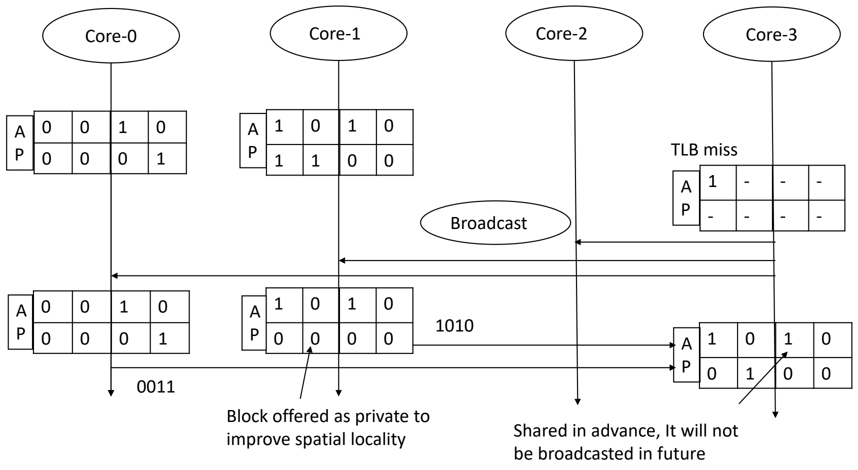

Core-0
Core-1
Core-2
Core-3
A
P
TLB miss
Broadcast
1010
0011
Block offered as private to improve spatial locality
Shared in advance, It will not be broadcasted in future

Figure 2: Block-grain Classification scheme

block with offset 0 in the depicted page. The *access bit* of the block with offset zero is set to 1 as the core is accessing the block. The request is broadcast to all TLBs in the system.

When core-0 receives the request from core-3, it makes the prediction of the cache blocks that it expects to use. Since it is not accessing blocks 0 and 1 the core will reply offering them as private (0 in the bit vector). Block 2 is already shared, so it will be not offered as private (1 in the bit vector). Finally, block 3 is potentially private at core-0, but not accessed. In this case, core 0 decides that, since it accessed block 2, and core-3 is asking for block 0, core-0 has more chances to access block 3 due to spatial locality and will not offer the block as private (1 in the bit vector). A simple OR operation between the access and private vector suffices to compute the response issued back to core-3. Core-0 sends a 0011 bit vector to core-3. Core-1 has block 0 as private, and it has to convert it to shared, thus resetting the private bit of block 0. Block 1 is this time offered as private by Core-1, since Core-3 is just accessing block 0, and spatial locality says that it may probably access block 1 in the near future. The general rule for spatial locality is that a core offer all blocks as private from the requested block to the first accessed block (block 1 in core-1 in this example).

Block 2 is shared, and therefore not offered as private, and block 3 can be offered as private. Core-1 sends a 1010 bit vector to core-3.

When core-3 receives all the replies, it performs a NOR operation for all replies (0100) to get the private bit vector. Additionally, core-3 can extract from the received vectors sharing information for blocks that it has not accessed. This happens in the example for block 2 and we call it the access prefetch optimization. In essence, when more than one TLB has claimed the use of a block, we can infer that the block will be a shared one when accessed. The way to mark it before the access as shared is to set it as accessed directly, even if it has not been accessed yet. This optimization allows to reduce the broadcast operation, due to classification misses, for shared blocks in the future. Both the spatial locality optimization and the access prefetch optimization help to reduce the number of broadcast operations in the block-grain scheme.

Protocol messages and their size. The TLB request now also contains the address of the requested block, not just the page address. The block address is used, for example, in the optimization for the spatial locality described in this section. As this information is commonly included in the header of the message, requests are made of 8 bytes. The responses issued by the TLBs include the usage information (access bit vector), which, considering that 4KB pages contain 64 blocks, is represented using 64 bits (8 bytes). In addition, some response messages may include the physical address translation, encoded using 4 bytes. Responses due to classification misses do not send the access bit vector nor the page translation information, thus using the 8-byte message format.

3.2.2. Storage requirements

Block-grain data classification stores the access and private bit vectors for the corresponding pages in TLB entries. Figure 3 shows the format of a TLB entry in the block-grain classification scheme (added fields in gray). As mentioned, two bit vectors are added to TLB entries: access and private. Accessed blocks are classified according to their private bit. Non-accessed blocks are considered

Virtual address	Physical address	Access bit vector					Private bit vector				
		0	0	0	0	0	1	1	1	1	1

Figure 3: TLB entry in Block-grain Classification

not to have any classification information if the private bit is 0, in which case the classification miss broadcast transaction triggers. If the private bit is 1, new accessed blocks are transparently (i.e., without informing other cores) moved to private. Therefore, only one core has a particular block with the private bit set to 1 independently from being accessed or not. The number of bits required by each vector is 64, the total number of blocks per page in 4KB pages. Therefore, each entry adds a total of 16 bytes, a number that does not grow with the number of cores in the system.

3.2.3. Adaptation to large pages

The majority of OS use a standard page size of 4KB since it allows greater granular control over the system. It is possible that in certain circumstances, huge pages may be supported, resulting in larger bit-vectors being used in the TLB, which can result in a significant increase in the storage needs. An alternative would be to consider large pages as shared by default, not performing extra optimizations in case they are employed [31].

3.3. Subpage-grain classification (SGC)

Finally, we propose a subpage-grain classification scheme aimed to achieve a trade-off between TLB storage and classification accuracy. This scheme works at a finer granularity than the page-grain policy but at a coarser granularity than the block-grain policy. The technique is based on the well-known concept of clustering, in a similar way as cores clustered in a coarse bit vector [33]. Without loss of generality, we assume in our implementation a clustering value K=4, that is, four contiguous blocks are represented using a single bit.

Shrinking the size of bit-vectors with clustering helps to reduce TLB storage requirements but also has an impact on performance. On the one hand, it may reduce the accuracy of classification, and on the other hand, it may reduce the number of TLB classification miss broadcast operations. We analyze this proposal along with page-grain and block-grain schemes.

3.3.1. Classification protocol

This classification protocol works similar to BGC. Broadcasting TLB requests happen in two situations: TLB miss and TLB classification miss. Again, the classification messages use the same network as other coherence messages. In case of a TLB miss, all TLBs reply with an access bit vector, where each bit represents a set of blocks in a cluster, instead of an individual block.

Subpage-grain classification has fewer TLB classification misses than the block-grain classification scheme as the classification of one block is used for its other three neighbours. The private-shared nature of the cluster of blocks is calculated after receiving all replies from the other core TLBs applying the NOR operation for all replies.

Protocol messages and their size. The subpage-grain classification protocol includes in the TLB requests the page address and the id of the cluster of blocks in the page accessed. This is again encoded in the header of the message. Each TLB core reply with a control message that includes a header with the target address (8 bytes). In case of a request generated by a TLB miss, and on a hit in the remote TLB, the response message adds the access bit vector (2 bytes now due to clustering, that is, four times less compared to information sent in the block-grain classification scheme) and the address translation (4 bytes). In case of a classification miss, the response just adds a single bit in the header indicating the use of the requesting cluster. SGC reduces therefore the size of the bit vector sent through the interconnect, thus helping to further reduce the traffic generated over the interconnection network.

3.3.2. Storage requirements

The subpage-grain classification technique requires adding two-bit vectors to each TLB entry (access and private), as the block-grain classification mech-

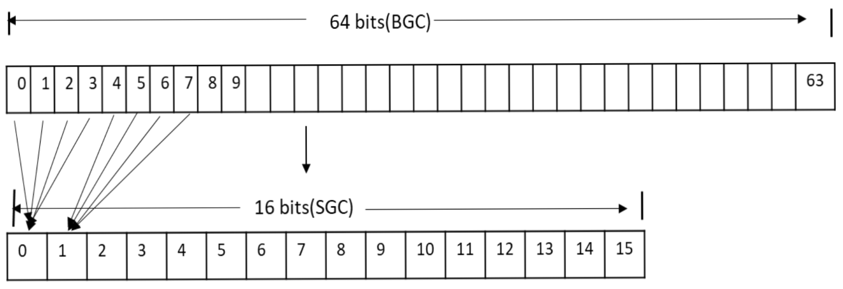


Figure 4: Subpage-grain Classification scheme

anism. Still, these vectors are four times smaller, reducing the storage area requirements. Figure 4 depicts the bit vector formation for the sub-page classification scheme, where the first bit in each vector corresponds to blocks 0, 1, 2, and 3 (and to the four first bits in a block-grain scheme). Overall the subpage-grain classification scheme adds 4 bytes to each TLB entry. The TLB scheme is similar to the one depicted for the block-grain scheme (see Figure 3).

3.3.3. Adaptation to large pages

As discussed in Section 3.2.3, large pages (e.g., 2MB) would enlarge the memory requirements of the access and private bit vectors, which can dramatically increase the memory requirements. The subpage-grain classification approach simply solves this problem by just modifying the K value. Larger pages can use a larger K value, just that the size of the vector is always the same, independent on the page.

4. Applying data classification to Token coherence

The Token coherence protocol broadcasts all cache miss requests without knowing the nature of the accessed block. In this work, we propose integrating an efficient data classification mechanism with a Token coherence protocol in order to filter such broadcast requests. This section discusses the integration of the Page-grain (PGC), Block-grain(BGC), and Subpage-grain (SGC) classification schemes in a Token coherence protocol.

On each memory reference, the core access in parallel both to the TLB and the L1 cache, since we assume virtually-indexed, physically tagged caches.[1]. In case of a cache miss, the TLB retrieves the classification for the accessed block: the one of the page, subpage, or block, depending on the implemented scheme.

If the classification for the block being accessed is shared, then the Token protocol behaviour is not altered. However, in case the classification is private, there is a guarantee that the current block is not accessed by other core, and issuing a broadcast request is therefore not required. In that case, the broadcast request is filtered, thus saving important network traffic.

Note that even if our classification mechanism relies on broadcast, once a block is classified as private, it can filter many Token broadcast requests due to cache misses as (1) the life of TLB pages is longer than the life of blocks in the local cache and (2) in a single TLB broadcast request our mechanisms can classify several blocks at the same time, even when considering the block-grain approach, since the protocol replies with the use information of all blocks within a page.

Figure 5 shows the classification integration with Token coherence. A request is broadcast to all TLBs in the case of TLB miss. BGC and SGC also perform the broadcast in the case of a TLB classification miss in parallel to the cache access and in case of a hit, the classification is not required. The requestor collects responses from all TLBs (vectors and page translation information) to form the private/shared classification. On a cache miss, Token coherence does not broadcast the coherence request for private blocks, but a single request to the home controller is sent to get the tokens and cache block. Otherwise, Token coherence performs the broadcast as usual.

[1]However, the mechanism can work with other cache designs [34]

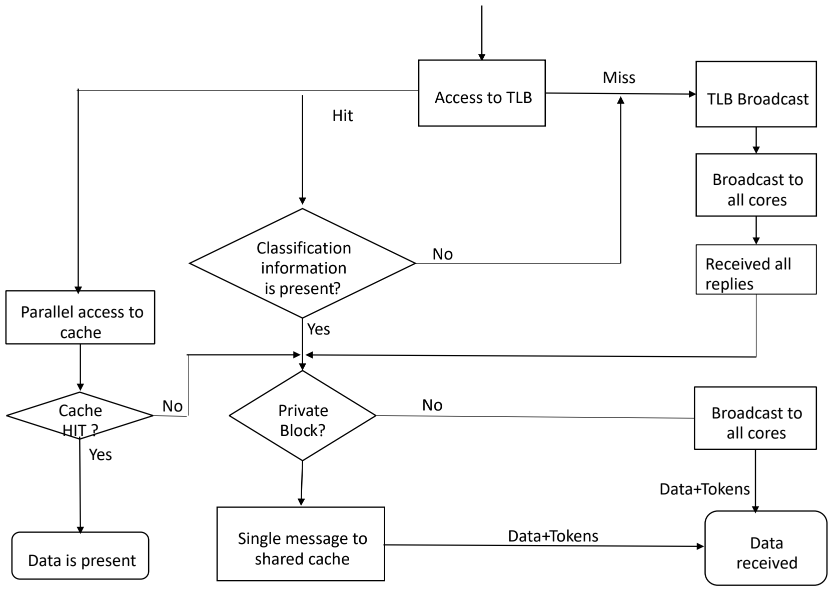


Figure 5: Token-grain classification scheme

4.1. Recovery mechanism

Commonly, when a block transitions from private to shared, a recovery mechanism is required as a block considered not coherent before becomes coherent now. For example, directory protocols require the eviction (or directory update [11]) of the blocks transitioning from private to shared as the directory does not track them. Token coherence protocols do not require the recovery process as the sharing information (tokens) is distributed among the coherence nodes. This, is an important simplification with respect to the integration of the classification in directory protocols and offers better performance for the classification as the eviction time can be quite large.

4.2. TLB evictions or TLB shoot-down

Private-shared data classification information is lost when a TLB entry is evicted due to TLB capacity or a TLB shoot-down. When invalidating a TLB entry, the usage information is lost, and the corresponding blocks for the page should not be locally cached. Otherwise, other TLB may believe that the block is not in use by other core and considered as private while indeed may be not private. All blocks belonging to the evicted page are invalidated from the cache to keep the classification consistent. This procedure is commonly supported when TLB shoot-downs take place, as the corresponding virtual address may have changed its physical location.

5. Simulation environment

We evaluate how the data classification at different granularities affects the performance of Token coherence. To this end, we perform full-system simulations using Virtutech Simics [35] and the Wisconsin GEMS [36] tool-set. We use the simple network model provided by GEMS. The simulated architecture is a tiled-CMP architecture (with 8(2x2), 16(4x4) and 32(4x8) tiles/cores) that maintains coherence using a Token protocol. We also model a directory-based coherence protocol to offer a comparison point between Token and directory protocols. We model in-order cores with private L1 caches and a shared unified L2 cache. Table 2 shows the parameters of the simulated system. Four memory references are considered for TLB miss latency to walk the page table, as in the 48-bit x86-64 virtual address space.

We apply the three classification schemes described in Section 3 to a Token protocol, and hence we refer to our resulting schemes as Token-PGC, Token-SGC, and Token-BGC. We refer to just Token to the baseline Token coherence protocol without a classification scheme. Token coherence uses persistent requests if normal communication reaches the time limit. Private-shared data classification is not applied to persistent requests to streamline the Token coherence.

These schemes are evaluated using 14 different parallel workloads from SPLASH-2 [37] and PARSEC [38]. Table 3 shows the input used for each benchmark. We report results for the parallel phase of each benchmark, and the classification mechanism is initialized at the beginning of the parallel phase. Due to the large

Table 2: System configuration

Processor	2.20GHz, 8, 16, and 32 cores.
Cache hierarchy	Non-inclusive. Token and directory-based coherence. L1 I/D caches: 64KB, 4-way (256 sets), 2 cycle access latency. Shared L2 cache: 1MB/tile, 8-way (2048 sets), 6 cycles access latency. Memory access time: 160 cycles.
Virtual memory	I/D TLB: 128 sets, 4 ways, 1 hit cycle, 1000 cycles to walk the page table, 4KB page sizes.
On-chip Network	4x4 2-D mesh with broadcast support. Flit size: 16 bytes. Bandwidth: 1 flit/cycle. 5-flit data and 1-flit control messages. 1-cycle switch and 1-cycle link time.

simulation time requirements of 32-core simulation, we selected the benchmarks that were able to be simulated to completion in a reasonable amount of time. In particular, we run the same benchmarks as in our previous work on classification [31], excluding Volrend since we noticed in our simulations the appearance of the performance bug described in the SPLASH-3 work [39].

6. Results

This section shows how different data classification schemes help to reduce the number of broadcast operations in Token coherence and, in the end, reduce the network traffic and improve system performance in Token coherence protocols.

6.1. Fraction of broadcast requests

Figure 6 shows the the percentage of broadcast requests for Token-PGC Coherence (First bar), Token-SGC Coherence (Second bar), and Token-BGC Coherence (Third bar) for 8 cores, 16 cores, and 32 cores. Each bar shows the broadcast requests split into coherence and classification. The baseline Token

Table 3: Benchmarks and input sizes

Benchmark	**Input size**
SPLASH-2 Benchmarks	
Barnes	8192 bodies, 4 time steps
Cholesky	tk15.O
FFT	64K complex doubles
FMM	16K particles
LU	512×512 matrix
LUNC	1024×1024 matrix, 64×64 blocks
Ocean	258×258 ocean
Radiosity	room, -ae 5000.0 -en 0.050 -bf 0.10
Radix	8,388,608 integers
Watersp	4096 molecules
PARSEC Benchmarks	
Blackscholes	Simmedium
Fluidanimate	Simsmall
Swaptions	Simsmall
x264	Simsmall

coherence contains only coherence broadcasts. Our private-shared data classification is able to filter the broadcast requests issued by the coherence protocol with minimal classification (TLB) broadcast request overhead, which ultimately leads to a decrease in the number of cache look-ups, and a consequent decrease in energy consumption.

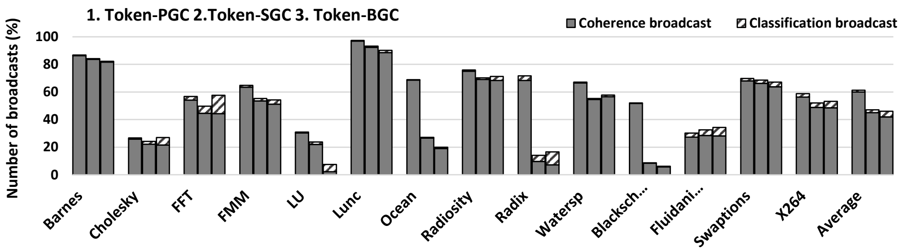
Figure 6a shows the the percentage of broadcast requests for the 8-core configuration. Token-PGC, working on a coarse granularity, classifies more memory blocks as shared, and the filtering is less effective. Indeed, Token-PGC reduces broadcast by 38.7% compared to the baseline Token coherence. Although Token-PGC coherence has lower memory overhead, its benefit is limited, and finer-grain classification is required to achieve larger reduction. Token-SGC improves the classification by allocating one bit to represent the classification of four blocks, and that helps reducing broadcast by 52.9% (14.2% reduction compared to Token-PGC). Token-BGC reduces the number of broadcast operations by 53.9%. The effectiveness of Token-SGC is therefore similar to Token-BGC (just 1.0% difference), and therefore it represents a good trade-off between storage requirements and filtering capabilities.

Figure 6b and Figure 6c focus on 16- and 32-core configurations, respectively. Token-PGC reduces broadcast by 26.4% and 21.3%, respectively. Token-SGC filters 40.1% and 30.0%, respectively, of broadcast requests. Finally, Token-BGC reduces even more broadcast operations (46.4% and 36.7%, respectively). As the number of core increases, the number of filtered broadcast operations is reducing as the sharing behavior of data increases as the number of core grows.

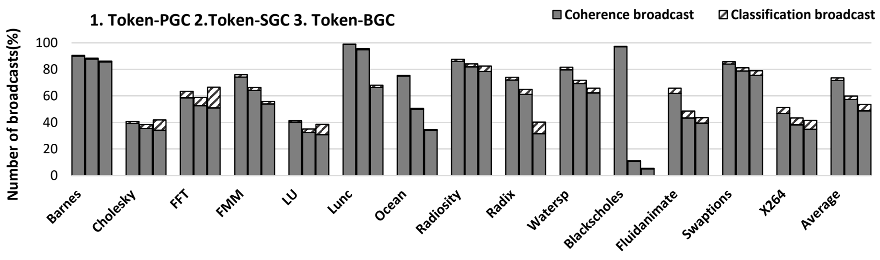
6.2. Normalized Network traffic

This section analyzes the overall impact on network traffic considering the TLB broadcast overhead in network traffic and the reduction in network traffic due to the broadcast filtering. Figure 7 shows the traffic in the on-chip network, flits transmitted from router to router, which it can be considered as a proxy for network energy consumption. The traffic is normalized with respect to Token protocol (first bar), and it is shown also for Directory (Second bar), Token-PGC (Third bar), Token-SGC (Fourth bar), and Token-BGC (Fifth bar) and for 8, 16 and 32 cores. Each bar differentiates the traffic based on cache request, cache response control, cache response data, TLB request control, TLB response control, and TLB response data. Token-SGC scheme and Token-BGC scheme have more TLB requests issued than the Token-PGC approach. This is because Token-PGC coherence has TLB broadcast only in one case, TLB miss. In contrast, Token-BGC and Token-SGC coherence broadcast in two instances: TLB miss and TLB classification miss, which increases the number of TLB broadcast operations.

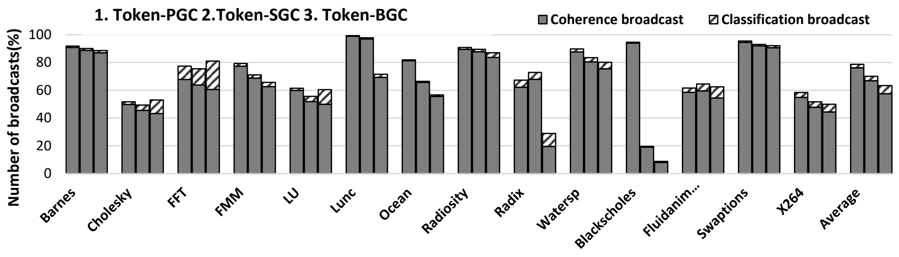
Directory, Token-PGC, Token-SGC, and Token-BGC reduce the network traffic by 6.9%, 11.3%, 15.3%, and 16.2%, respectively for 8 cores. Token-SGC



(a) 8 cores



(b) 16 cores

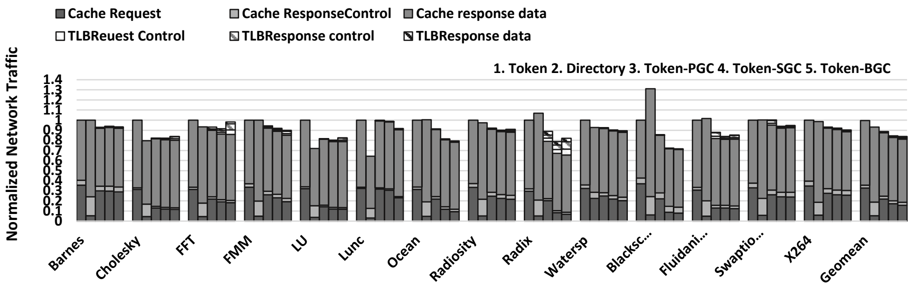


(c) 32 cores

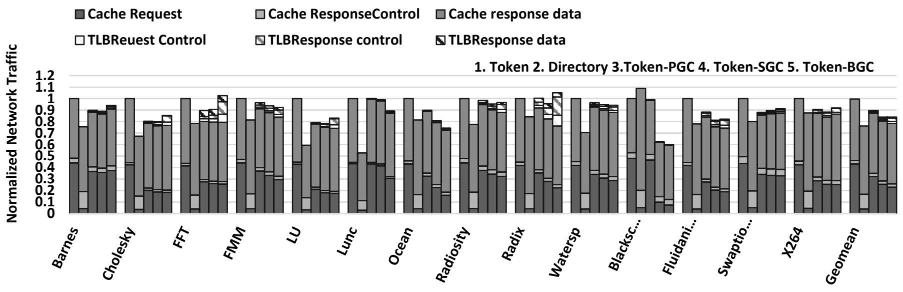
Figure 6: Number of broadcast requests in Token coherence

and Token-BGC reduce the network traffic by 4.0% and 5.0% compared to the Token-PGC coherence scheme. The reductions are in line with Figure 6a. For example, Blackscholes is the application that reduces more broadcast requests, as it is highly parallel, and it therefore reduces to a greater extent the cache request traffic.

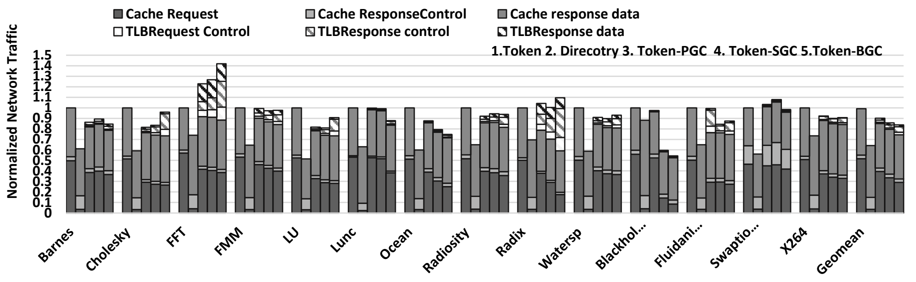
For 16 cores, on average, the overhead of the number of TLB requests in Token-SGC and Token-BGC is 1.5% and 2.7%, respectively, compared to the



(a) 8 cores



(b) 16 cores



(c) 32 cores

Figure 7: Normalized Network traffic

Token-PGC approach. FFT and Radix show more TLB communication overhead in Token-BGC, since pages are densely accessed and with irregular sharing patterns. Token-SGC helps to reduce the TLB communication overhead that is observed in the FFT, Radix benchmarks, at the cost of classification accuracy. Directory, Token-PGC, Token-SGC, and Token-BGC reduce the network traffic by 24.0%, 10.3%, 16.0%, and 16.1% than the baseline Token protocol for

16 cores. On average, Token-BGC reduces network traffic by 6.0% compared to Token-PGC. Token-SGC reduces the TLB miss classification broadcast compared to the Token-BGC, which helps to reduce the network traffic. At the same time, Token-BGC has more reduction in the broadcast operation of private data. They both compensate each other and turns into the same performance on average.

For 32 cores, Directory, Token-PGC, Token-SGC, and Token-BGC reduce the network traffic by 35.9%, 9.8%, 14.4%, and 16.2% compared to baseline Token coherence. Ocean and Blackscholes shows more broadcast reduction in Figure 6c) that reflects in more reduction in the network traffic for 32 cores. FFT and Radix benchmark increase the overall network traffic as it has more TLB communication network traffic. FFT and Radix spend increase network traffic in TLB communication for tracking private data, but at the same time the TLB communication helps to find address translation and to accelerate TLB misses. We can observe that as the number of core increases, the TLB broadcast increases. Token-SGC reduces the same network traffic compared to Token-BGC.

6.3. L1 cache miss latency

Figure 8 shows the average L1 cache miss latency in Token coherence (First bar), Directory (Second bar), Token-PGC (Third bar), Token-SGC (Fourth bar), and Token-BGC (Fifth bar) for 8 cores, 16 cores, and 32 cores.

For 8 cores, the L1 cache miss latency in Token, Directory, Token-PGC, Token-SGC, and Token-BGC is 64.1, 70.0, 47.9, 48.1, and 49.0 cycles, respectively. FFT and Radix show around 57.4% and 44.5% reduction in cycles for all the classification schemes compared to baseline Token coherence. For 16 cores, the L1 cache miss latency in Token, Directory, Token-PGC, Token-SGC, and Token-BGC is 57.6, 63.7, 40.0, 40.5, and 41.5 cycles, respectively. FFT and Radix show around 35.3% and 46.5% reduction in cycles for all the three classification schemes compared to baseline Token coherence. For 32 cores, the L1 cache miss latency in Token, Directory, Token-PGC, Token-SGC, and Token-BGC is 75.8, 82.9, 50.6, 49.2, and 50.9 cycles, respectively.

Token coherence reduces cache miss latency by avoiding the indirection to the directory present in directory protocols. Our classification schemes applied to Token coherence still include the advantage of avoiding indirection, but also reduce broadcast and accelerate address translation. By reducing broadcast requests, the latency of cache misses is also reduced. As the number of cores increase, the reduction in L1 cache miss latency also increases. On the other hand, the increase in traffic due to TLB communication does not negatively affect cache miss latency, as this process is done before the cache miss and also is done in parallel to the page table walking, which is usually the limiting factor on a TLB miss.

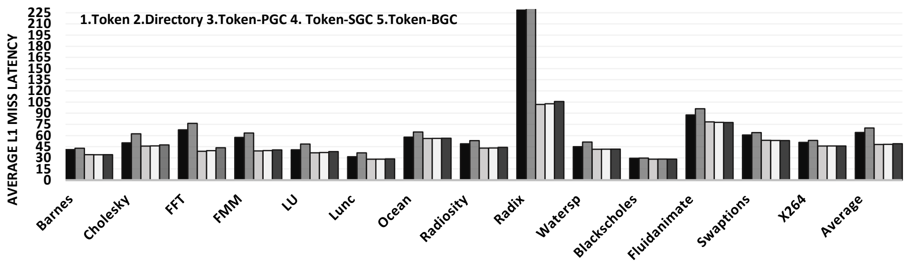
6.4. Execution time

Figure 9 presents the execution time for Directory (Second bar), Token-PGC (Third bar), Token-SGC (Fourth bar), and Token-BGC (Fifth bar), normalized with respect to baseline Token coherence (First bar), for 8, 16, and 32 cores.

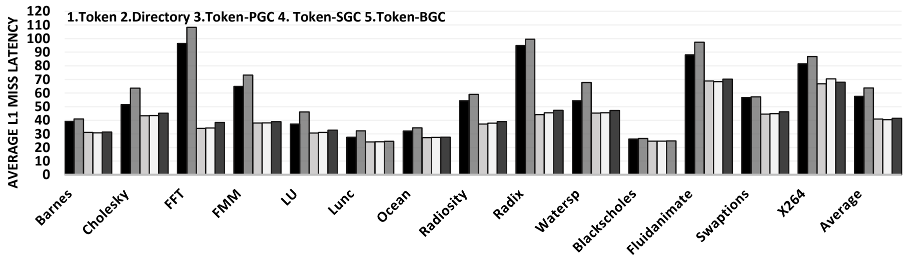
The first observation is that Token and Directory perform practically on par. Directory requires access to the home tile for every cache miss, resulting in most cache misses requiring more time to obtain the desired data. This latency increases as the number of cores increase. On the other hand, Token, although it avoids the indirection to the home node, relies on broadcast. The broadcast also becomes more problematic as the number of cores increase. Hence, the similar results obtained for both protocols.

As mentioned, there are several factors that make classification mechanisms faster. First, they reduce broadcast at the same time that benefit from the lack of indirection of Token. Additionally, they accelerate page translation thanks to the TLB-to-TLB communication [14], which is one of the key reasons behind the performance improvements.

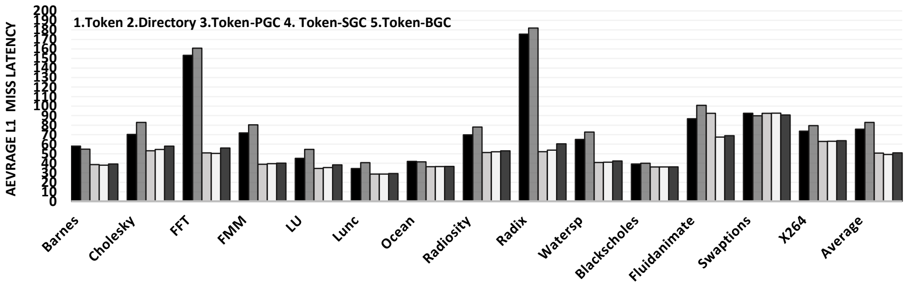
In particular, when considering 8 cores (Figure 9a), the execution time improvement in Token-PGC, Token-SGC, and Token-BGC is 10.6%, 10.7%, 10.5%



(a) 8 cores



(b) 16 cores



(c) 32 cores

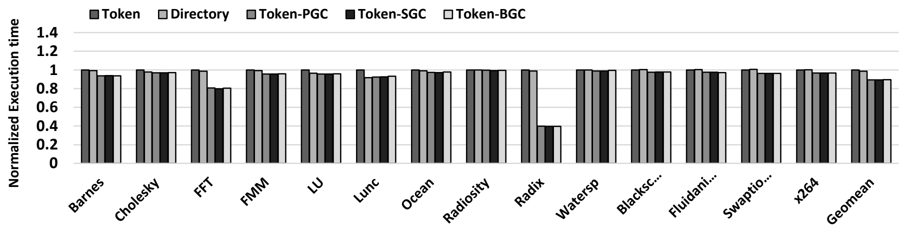
Figure 8: L1 cache miss latency

respectively, with respect to the baseline token coherence protocol. Directory protocol gives almost the same performance as Token protocol. Radix has a maximum reduction in the L1 cache miss latency (44.5%), as seen in Figure 8, which results in a 41.0% improvement in all three schemes. It uses more TLB communication traffic to determine classification, but it also lowers the number of cycles to perform the page address translation. Note that as mentioned previously, this extra traffic is in parallel with the address translation, which is the factor limiting the latency on a TLB miss. When considering 16 cores (Figure 9b), Directory, Token-PGC, Token-SGC, and Token-BGC reduce the execution time by 1.0%, 12.3%, 12.0%, and 12.9% compared to the baseline Token protocol. When considering 32 cores (Figure 9c), Directory, Token-PGC, Token-SGC, and Token-BGC reduce the execution time by 2.0%, 25.7%, 25.3%, and 25.2%, respectively, with respect to the baseline token protocol. The performance improvement is doubled for 32 cores compared to the 16-core configuration. The classification provides more performance benefits as the number of cores increases in the system.

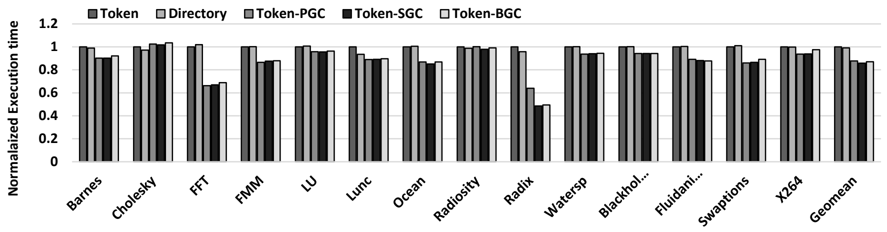
6.5. Storage overhead

This section compares the storage overhead of the three classification schemes analyzed and the two cache coherence protocols considered. Coherence protocols entail extra overhead to track the coherence information. The classification mechanism adds extra overhead due to storing the classification information.

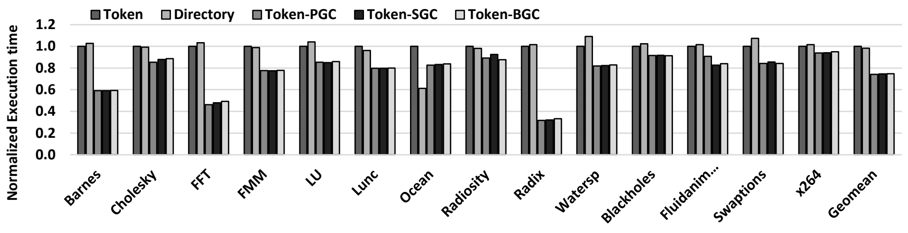
Table 4 shows the memory overhead (in KB) of the coherence protocols for each core in the system, which depends on the number of cores. Token coherence keeps the token count for any block stored in the L1 and L2 caches, requiring 1+log2(n) bits (the owner-token bit and non-owner token count) –the L1 and L2 cache level stores these extra bits in the tag part of the caches. Our directory-based protocol stores the directory information either in the L2 tags, when the L2 cache holds a copy of the block, or in a distributed directory cache, when the block is stored in any of the L1 caches but not in the L2 cache. Therefore, no invalidation messages due to directory evictions are generated. In our directory-based protocol the information is stored using a full-map sharing code. Hence, the number of bits required corresponds to the number of cores in the system. Different from the L2, where the sharing information is added to the tags, the directory is a standalone cache, and therefore we account for the tag (32 bits) associated to the sharing codes. The directory has a coverage of 100% with respect to the L1 caches, that is, the same number of entries.



(a) 8 cores



(b) 16 cores



(c) 32 cores

Figure 9: Normalized execution time

Table 4: Per-core storage overhead of coherence protocols

			8 cores		**16 cores**		**32 cores**	
Protocol	**Structure**	**Entries**	**Entry size (bits)**	**Total size (KB)**	**Entry size (bits)**	**Total size (KB)**	**Entry Size (bits)**	**Total size (KB)**
Token	L1D tags	1K	4	0.5	5	0.625	6	0.75
	L1I tags	1K	4	0.5	5	0.625	6	0.75
	L2 tags	16K	4	8	5	10	6	12
Directory	L2 tags	16K	8	16	16	32	32	64
	Directory	2K	32+8	10	32+16	12	32+32	16

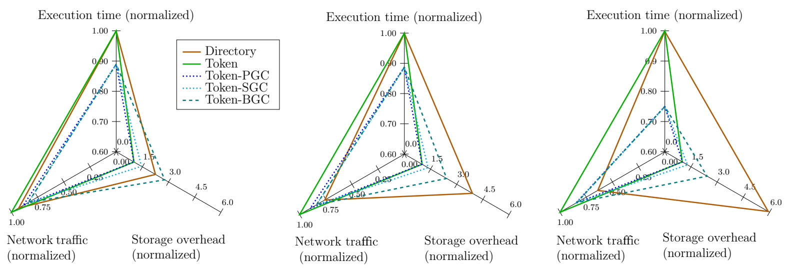


Figure 10: Trade-off among the main design goals for coherence protocols

Table 5 shows the memory overhead (in KB) of the classification schemes for each core in the system, which does not depend on the number of cores in the system. PGC adds 1 bit for private-shared page information per TLB entry. BGC classification adds two 64-bit bit-vectors per TLB entry, reaching an overhead of 16KB per core. SGC classification reduces by four the overhead of BGC classification, being just 4KB per core. These classification methods do not increase the area overhead as the number of cores increases. On the contrary, storage demands of Token grow logarithmically as the number of cores increases, and worse, storage demands of a full-map directory grow linearly with the number of cores. The memory requirements for Token-PGC, Token-SGC, and Token-BGC correspond to the addition of the memory requirements of both the Token protocol and the classification protocol.

Table 5: Per-core storage overhead of classification protocols

Protocol	**Structure**	**Entries**	**Entry size (bits)**	**Total size (KB)**
PGC	ITLB	512	1	0.0625
	DTLB	512	1	0.0625
SGC	ITLB	512	32	2
	DTLB	512	32	2
BGC	ITLB	512	128	8
	DTLB	512	128	8

6.6. Overall analysis

Figure 10 shows a three-fold trade-off among network traffic, execution time, and storage overhead for the evaluated coherence and classification protocols. The three-axis are normalized with respect to a Token protocol. Again, we show three configurations: 8 cores, 16 cores, and 32 cores. In general, the Token coherence protocol does not achieve good results for all the parameters. The Token protocol has the highest network traffic and execution time, but it also has fewer memory requirements. In contrast, the directory protocol requires the largest memory requirements and the lowest network traffic in all protocols. For an 8-core configuration, Directory, Token-PGC, Token-SGC, and Token-BGC can reduce network traffic by 6.0%, 12.0%, 15.0%, and 16.0% compared to baseline directory protocol, respectively. Compared to baseline Token protocol, Directory, Token-PGC, Token-SGC, and Token-BGC reduce execution time by 1.0%, 9.0%, 11.0%, and 10.0%, respectively.

Although Token coherence has acceptable storage requirements for a 16-core configuration, it is limited by traffic. Token-PGC has a 12.0% execution time improvement compared to the Token protocol, and also requires lower storage. Token-PGC experiences more network traffic than the Token-SGC and Token-BGC protocols. Compared to a Directory protocol, Token-SGC can better compromise network traffic with less memory requirements and still guarantee low average execution time (14.0% improvements). Token-PGC has only 1.0% more memory requirement compared to the Token protocol.

For a 32-core configuration, all classification-based protocols have fewer memory requirements than the directory protocol and a 24.0% improvement in execution time. As the number of core grows, the TLB communication overhead can increase. However, the classification protocols manage to reduce this overhead considerably.

In general, the baseline Token coherence protocol and Directory protocol do not get good results for all the performance metrics. However, shrinking bit vectors for Token-SGC can lead to a good compromise between memory requirements and network traffic, improving execution time.

7. Related work

Classifying data as private or shared can improve the energy efficiency of the cache memories [40, 41], remove interference misses by allowing cache partitioning [42], and improve the cache coherence mechanism, leading to system performance [15, 19, 43, 25, 26, 27, 44, 45, 31]. In particular, some works aim to reduce access latency to NUCA caches in a tiled-CMP [46, 20], other works to reduce the directory size [11, 32], and other works to bring down the number of broadcast operations in snoop-based cache coherence protocols [4, 47, 48].

There are several compiler, hardware, and operating system-based techniques to find private-shared data classification at different granularity. In a compiler-dependent approach [20, 12], it is hard to know during compilation time what will be the status of a variable at execution time. Our proposed schemes work during program execution which gives more precise classification. Hardware-based techniques implemented at the coherence directory [8, 22, 26] work on finer granularity, but they entail large storage requirements. More importantly, the classification is unknown in directory-based schemes until the directory is accessed, thus preventing their use to filter coherence traffic in snoop-based protocols. Operating-system-based data classification mechanisms [46, 11, 13] use the existing hardware structure like page-table and TLB and, as a result, they do not require important hardware changes. However, operating system-based data classifications work at a coarse (page) granularity, resulting in the misclassification of blocks in the presence of false sharing within their page.

Achieving accurate fine-grain classification before a cache miss takes place has been previously investigated. Davari et al. [43] made a potential study on both granularity and adaptivity, showing a potential for both factors in a number of applications. Upadhyay et al. [15, 31] proposed a block grain classification to improve directory-based coherence. However, it comes with some extra storage at the TLB level, which we minimize in this work. Also, we apply the resulting classification to filtering traffic in Token protocols, showing potential for that kind of protocol too. Soltaniyeh et al. [32] analyzed a subpage-grain data classification to improve directory-based protocols. However, their alternative employs an on-chip page table and requires modification in the operating system. In addition, their page tables store the keeper information for each subpage and the other information, which results in storage overhead to hold all keepers. Differently from their subpage-grain data classification, our proposed SGC classification leverages TLB-to-TLB communication and reduces storage requirements.

Subspace snooping [4] also targets broadcast filtering in snooping-based protocols. Their approach is to store the sharer information in the page table and converts the broadcast request into a multicast only to the sharer of the block. Therefore, they require to store information about the sharers of a block. In contrast, our approach uses a single bit indicating the private or shared nature of a page, which is more efficient. More importantly, they can just support page granularity, while our approach can go beyond the page limit.

Coarse-Grain Coherence Tracking [21] monitors the coherence status of large memory regions and uses that information to avoid unnecessary broadcasts in a multiprocessor system. Differently from coarse-grain coherence tracking, our classification techniques work at the TLB level. This brings several benefits, such as fast address translation, classification provided before the first cache miss, more effective, and reusing TLB tags, thus requiring less storage than coarse-grain coherence tracking.

TokenTLB [24, 27] is a novel classification technique using TLB structure directly to reduce consumption in directory cache coherence protocols. TokenTLB works like Token coherence [16], applying tokens for classification as it does not require issuing persistent requests as in the case of races, the page is classified as shared. They are also restricted to page granularity, although that technique could be combined with our SGC to improve accuracy. However, the storage requirement of tracking token at subpage granularity would be much higher.

The proposed schemes work on different granular approaches and analyze the trade of area and token cache coherence performance using the TLB broadcast mechanism.

8. Conclusion

We have proposed to apply fine-grain private/shared data classification techniques to filter broadcast traffic in Token coherence. The described classification mechanisms work at the TLB level and provide the private/shared classification for a missing block in the cache before any coherence action to obtain that block is performed. This allows us to filter unnecessary broadcast requests for private data in Token coherence and reduce the L1 cache miss latency.

We have analyzed the trade-off among classification storage, performance, and traffic, showing that a subpage-grain classification offers a good compromise among them. Token-SGC requires less area storage and performs similar to Token-BGC. The proposed Token-SGC scheme eliminates, on average, 40.1% of broadcast request operations and results in a reduction of network traffic by 16.0% and a performance improvement of 12.0% compared to the baseline Token protocol for 16-cores. In terms of storage requirements, Token-SGC adds a moderate amount of storage requirement. Token-SGC has a bit more storage requirement than the baseline Token coherence protocol and four times less than the Token-BGC protocol.

ACKNOWLEDGMENTS

This work was supported by the Spanish MCIU and AEI, as well as European Commission FEDER funds, under the grant "RTI2018-098156-B-C53".

References

[1] D. E. Culler, J. P. Singh, A. Gupta, Parallel Computer Architecture: A Hardware/Software Approach, Morgan Kaufmann Publishers, Inc., 1999.

[2] J. F. Cantin, J. E. Smith, M. H. Lipasti, A. Moshovos, B. Falsafi, Coarse-grain coherence tracking: Regionscout and region coherence arrays, IEEE Micro 26 (1) (2006) 70–79. `doi:10.1109/MM.2006.8`.

[3] N. Agarwal, L. Peh, N. K. Jha, In-network coherence filtering: snoopy coherence without broadcasts, in: 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-42 2009), ACM, 2009, pp. 232–243. `doi:10.1145/1669112.1669143`.

[4] D. Kim, J. Ahn, J. Kim, J. Huh, Subspace snooping: filtering snoops with operating system support, in: 19th International Conference on Parallel Architectures and Compilation Techniques, PACT, ACM, 2010, pp. 111–122. `doi:10.1145/1854273.1854292`.

[5] D. Kim, H. Kim, J. Huh, Virtual snooping: Filtering snoops in virtualized multi-cores, in: 43rd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO, IEEE Computer Society, 2010, pp. 459–470. `doi:10.1109/MICRO.2010.16`.

[6] M. Ferdman, P. Lotfi-Kamran, K. Balet, B. Falsafi, Cuckoo directory: A scalable directory for many-core systems, in: 17th International Conference on High-Performance Computer Architecture (HPCA-17), IEEE Computer Society, 2011, pp. 169–180. `doi:10.1109/HPCA.2011.5749726`.

[7] D. Sánchez, C. Kozyrakis, SCD: A scalable coherence directory with flexible sharer set encoding, in: 18th International Symposium on High Performance Computer Architecture, HPCA, IEEE Computer Society, 2012, pp. 129–140. `doi:10.1109/HPCA.2012.6168950`.

[8] J. Zebchuk, B. Falsafi, A. Moshovos, Multi-grain coherence directories, in: The 46th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO-46, ACM, 2013, pp. 359–370. `doi:10.1145/2540708.2540739`.

[9] L. G. Menezo, V. Puente, J. Gregorio, Flask coherence: A morphable hybrid coherence protocol to balance energy, performance and scalability, in: 21st IEEE International Symposium on High Performance Computer Architecture, HPCA, IEEE Computer Society, 2015, pp. 198–209. `doi:10.1109/HPCA.2015.7056033`.

[10] M. M. K. Martin, M. D. Hill, D. A. Wood, Token coherence: Decoupling performance and correctness, in: 30th International Symposium on Computer Architecture (ISCA), IEEE Computer Society, 2003, pp. 182–193. doi:10.1109/ISCA.2003.1206999.

[11] B. Cuesta, A. Ros, M. E. Gómez, A. Robles, J. Duato, Increasing the effectiveness of directory caches by avoiding the tracking of noncoherent memory blocks, IEEE Transactions on Computers 62 (3) (2013) 482–495. doi:10.1109/TC.2011.241.

[12] Y. Li, R. G. Melhem, A. K. Jones, Practically private: enabling high performance CMPs through compiler-assisted data classification, in: International Conference on Parallel Architectures and Compilation Techniques, PACT, ACM, 2012, pp. 231–240. doi:10.1145/2370816.2370852.

[13] A. Ros, S. Kaxiras, Complexity-effective multicore coherence, in: International Conference on Parallel Architectures and Compilation Techniques, PACT, ACM, 2012, pp. 241–252. doi:10.1145/2370816.2370853.

[14] A. Ros, B. Cuesta, M. E. Gómez, A. Robles, J. Duato, Temporal-aware mechanism to detect private data in chip multiprocessors, in: 42nd International Conference on Parallel Processing, ICPP, IEEE Computer Society, 2013, pp. 562–571. doi:10.1109/ICPP.2013.70.

[15] B. R. Upadhyay, A. Ros, M. NS, TLB-based block-grain classification of private data, in: 28th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, PDP, IEEE, 2020, pp. 122–130. doi:10.1109/PDP50117.2020.00025.

[16] M. R. Marty, J. D. Bingham, M. D. Hill, A. J. Hu, M. M. K. Martin, D. A. Wood, Improving multiple-CMP systems using token coherence, in: 11th International Conference on High-Performance Computer Architecture (HPCA-11), IEEE Computer Society, 2005, pp. 328–339. doi:10.1109/HPCA.2005.17.

[17] S. Shukla, M. Chaudhuri, Tiny directory: Efficient shared memory in many-core systems with ultra-low-overhead coherence tracking, in: International Symposium on High Performance Computer Architecture, HPCA, IEEE Computer Society, 2017, pp. 205–216. doi:10.1109/HPCA.2017.24.

[18] A. Esteve, A. Ros, M. E. Gómez, A. Robles, J. Duato, TLB-based temporality-aware classification in CMPs with multilevel TLBs, IEEE Transactions on Parallel and Distributed Systems 28 (8) (2017) 2401–2413. doi:10.1109/TPDS.2017.2658576.

[19] P. Caheny, L. Alvarez, M. Valero, M. Moretó, M. Casas, Runtime-assisted cache coherence deactivation in task parallel programs, in: the International Conference for High Performance Computing, Networking, Storage, and Analysis, SC, IEEE / ACM, 2018, pp. 35:1–35:12. doi:10.1109/SC.2018.00038.

[20] Y. Li, A. Abousamra, R. G. Melhem, A. K. Jones, Compiler-assisted data distribution for chip multiprocessors, in: 19th International Conference on Parallel Architectures and Compilation Techniques, PACT, ACM, 2010, pp. 501–512. doi:10.1145/1854273.1854335.

[21] J. F. Cantin, M. H. Lipasti, J. E. Smith, Improving multiprocessor performance with coarse-grain coherence tracking, in: 32nd International Symposium on Computer Architecture (ISCA), IEEE Computer Society, 2005, pp. 246–257. doi:10.1109/ISCA.2005.31.

[22] S. H. Pugsley, J. B. Spjut, D. W. Nellans, R. Balasubramonian, SWEL: hardware cache coherence protocols to map shared data onto shared caches, in: 19th International Conference on Parallel Architectures and Compilation Techniques, PACT, ACM, 2010, pp. 465–476. doi:10.1145/1854273.1854331.

[23] H. Zhao, A. Shriraman, S. Dwarkadas, V. Srinivasan, SPATL: honey, I shrunk the coherence directory, in: International Conference on Parallel

Architectures and Compilation Techniques, PACT, IEEE Computer Society, 2011, pp. 33–44. doi:10.1109/PACT.2011.10.

[24] A. Esteve, A. Ros, A. Robles, M. E. Gómez, TokenTLB+CUP: A token-based page classification with cooperative usage prediction, IEEE Transactions on Parallel and Distributed Systems 29 (5) (2018) 1188–1201. doi:10.1109/TPDS.2017.2782808.

[25] N. Ho, I. I. Ashraf, P. Kaufmann, M. Platzner, Accurate private/shared classification of memory accesses: A run-time analysis system for the LEON3 multi-core processor., in: Design, Automation & Test in Europe Conference & Exhibition (DATE), IEEE, 2017, pp. 788–793. doi:10.23919/DATE.2017.7927096.

[26] H. Hossain, S. Dwarkadas, M. C. Huang, POPS: coherence protocol optimization for both private and shared data, in: International Conference on Parallel Architectures and Compilation Techniques, PACT, IEEE Computer Society, 2011, pp. 45–55. doi:10.1109/PACT.2011.11.

[27] A. Esteve, A. Ros, A. Robles, M. E. Gómez, J. Duato, TokenTLB: A token-based page classification approach, in: International Conference on Supercomputing (ICS), ACM, 2016, pp. 26:1–26:13. doi:10.1145/2925426.2926280.

[28] B. F. Romanescu, A. R. Lebeck, D. J. Sorin, A. Bracy, Unified instruction/translation/data (UNITD) coherence: One protocol to rule them all, in: 16th International Conference on High-Performance Computer Architecture (HPCA-16), IEEE Computer Society, 2010, pp. 1–12. doi:10.1109/HPCA.2010.5416643.

[29] S. Srikantaiah, M. T. Kandemir, Synergistic TLBs for high performance address translation in chip multiprocessors, in: 43rd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO, IEEE Computer Society, 2010, pp. 313–324. doi:10.1109/MICRO.2010.26.

[30] D. Lustig, A. Bhattacharjee, M. Martonosi, TLB improvements for chip multiprocessors: Inter-core cooperative prefetchers and shared last-level TLBs, ACM Transactions on Architecture and Code Optimization (TACO) 10 (1) (2013) 2:1–2:38. `doi:10.1145/2445572.2445574`.

[31] B. R. Upadhyay, A. Ros, J. Shah, Efficient classification of private memory blocks, Journal of Parallel and Distributed Computing 157 (2021) 256–268. `doi:10.1016/j.jpdc.2021.07.005`.

[32] M. Soltaniyeh, I. Kadayif, O. Ozturk, Classifying data blocks at subpage granularity with an on-chip page table to improve coherence in tiled CMPs, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 37 (4) (2018) 806–819. `doi:10.1109/TCAD.2017.2729280`.

[33] A. Gupta, W.-D. Weber, T. Mowry, Reducing memory and traffic requirements for scalable directory-based cache coherence schemes, in: Scalable shared memory multiprocessors, Springer, 1992, pp. 167–192. `doi:10.1007/978-1-4615-3604-8_9`.

[34] S. Kaxiras, A. Ros, A new perspective for efficient virtual-cache coherence, in: 40th Annual International Symposium on Computer Architecture, ACM, 2013, pp. 535–546. `doi:10.1145/2485922.2485968`.

[35] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hållberg, J. Högberg, F. Larsson, A. Moestedt, B. Werner, Simics: A full system simulation platform, Computer 35 (2) (2002) 50–58. `doi:10.1109/2.982916`.

[36] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, D. A. Wood, Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset, ACM SIGARCH Computer Architecture News 33 (4) (2005) 92–99. `doi:10.1145/1105734.1105747`.

[37] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, A. Gupta, The SPLASH-2 programs: Characterization and methodological considerations, in: 22nd

Annual International Symposium on Computer Architecture, ISCA, ACM, 1995, pp. 24–36. doi:10.1145/223982.223990.

[38] C. Bienia, S. Kumar, J. P. Singh, K. Li, The PARSEC benchmark suite: characterization and architectural implications, in: 17th International Conference on Parallel Architectures and Compilation Techniques, PACT, ACM, 2008, pp. 72–81. doi:10.1145/1454115.1454128.

[39] C. Sakalis, C. Leonardsson, S. Kaxiras, A. Ros, Splash-3: A properly synchronized benchmark suite for contemporary research, in: International Symposium on Performance Analysis of Systems and Software, ISPASS, IEEE Computer Society, 2016, pp. 101–111. doi:10.1109/ISPASS.2016.7482078.

[40] J. J. Valls, A. Ros, M. E. Gómez, J. Sahuquillo, The tag filter architecture: An energy-efficient cache and directory design, Journal of Parallel and Distributed Computing 100 (2017) 193–202. doi:10.1016/j.jpdc.2016.04.016.

[41] B. R. Upadhyay, T. S. B. Sudarshan, Low power predictive placement cache scheme for embedded system, in: 2014 International Conference on Embedded Systems (ICES), 2014, pp. 250–254. doi:10.1109/EmbeddedSys.2014.6953167.

[42] J. J. Valls, M. E. Gómez, A. Ros, J. Sahuquillo, A directory cache with dynamic private-shared partitioning, in: 23rd IEEE International Conference on High Performance Computing, HiPC, IEEE Computer Society, 2016, pp. 382–391. doi:10.1109/HiPC.2016.051.

[43] M. Davari, A. Ros, E. Hagersten, S. Kaxiras, The effects of granularity and adaptivity on private/shared classification for coherence, ACM Transactions on Architecture and Code Optimization (TACO) 12 (3) (2015) 26:1–26:21. doi:10.1145/2790301.

[44] N. Parvathy, B. R. Upadhyay, T. S. B. Sudarshan, Cache coherence: A walkthrough of mechanisms and challenges, in: International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT), 2016, pp. 2251–2256. doi:10.1109/ICEEOT.2016.7755093.

[45] B. R. Upadhyay, T. Sudarshan, Task-enabled instruction cache partitioning scheme for embedded system, in: First International Conference on Smart System, Innovations and Computing, Springer, 2018, pp. 603–612. doi:10.1007/978-981-10-5828-8_58.

[46] N. Hardavellas, M. Ferdman, B. Falsafi, A. Ailamaki, Reactive NUCA: near-optimal block placement and replication in distributed caches, in: 36th International Symposium on Computer Architecture (ISCA), ACM, 2009, pp. 184–195. doi:10.1145/1555754.1555779.

[47] F. Yuan, Z. Ji, DP&TB: a coherence filtering protocol for many-core chip multiprocessors, The Journal of Supercomputing 66 (1) (2013) 249–261. doi:10.1007/s11227-013-0900-4.

[48] M. Ekman, P. Stenström, F. Dahlgren, TLB and snoop energy-reduction using virtual caches in low-power chip-multiprocessors, in: International Symposium on Low Power Electronics and Design, ACM, 2002, pp. 243–246. doi:10.1145/566408.566471.