# Boosting Performance of Directory-based Cache Coherence Protocols with Coherence Bypass at Subpage Granularity and A Novel On-chip Page Table

Mohammadreza Soltaniyeh [*]
Rutgers University
m.soltaniyeh@cs.rutgers.edu

Ismail Kadayif
Canakkale Onsekiz Mart University
kadayif@comu.edu.tr

Ozcan Ozturk
Bilkent University
ozturk@cs.bilkent.edu.tr

## ABSTRACT

Chip multiprocessors (CMPs) require effective cache coherence protocols as well as fast virtual-to-physical address translation mechanisms for high performance. Directory-based cache coherence protocols are the state-of-the-art approaches in many-core CMPs to keep the data blocks coherent at the last level private caches. However, the area overhead and high associativity requirement of the directory structures may not scale well with increasingly higher number of cores.

As shown in some prior studies, a significant percentage of data blocks are accessed by only one core, therefore, it is not necessary to keep track of these in the directory structure. In this study, we have two major contributions. First, we show that compared to the classification of cache blocks at page granularity as done in some previous studies, data block classification at subpage level helps to detect considerably more private data blocks. Consequently, it reduces the percentage of blocks required to be tracked in the directory significantly compared to similar page level classification approaches. This, in turn, enables smaller directory caches with lower associativity to be used in CMPs without hurting performance, thereby helping the directory structure to scale gracefully with the increasing number of cores. Memory block classification at subpage level, however, may increase the frequency of the Operating System's (OS) involvement in updating the maintenance bits belonging to subpages stored in page table entries, nullifying some portion of performance benefits of subpage level data classification. To overcome this, we propose a distributed on-chip page table as a our second contribution.

## Keywords

cache coherence; directory cache; many-core system; virtual memory; page table

[*]This work was done while the author was at Bilkent University.

*CF'16, May 16-19, 2016, Como, Italy*
© 2016 ACM. ISBN 978-1-4503-4128-8/16/05…$15.00
DOI: http://dx.doi.org/10.1145/2903150.2903175

## 1. INTRODUCTION

Effective cache coherence protocols as well as fast virtual-to-physical address translation are both critical to meet the demand for high-performance systems.

Directory-based cache coherence protocols are the state-of-the-art approaches in many-core CMPs to keep the data blocks coherent at the last level private caches. However, the area overhead, high energy consumption, and high associativity requirement of the directory structures may not scale well with increasingly higher number of cores in a chip. Different directory organizations have been suggested to enable lower overhead and higher scalability in coherence protocols relying on directories [9, 14].

To enable fast virtual memory translation, recent studies focused on designing Translation Look-aside Buffers (TLBs). There are different studies in the literature which try to propose efficient TLB organization, particularly for CMPs [5, 17]. We will discuss the drawbacks of those architectures and then propose our idea to enhance the performance of virtual to physical address translation in CMPs in the rest of the paper.

Several prior studies [10, 13] suggested page granularity data classification mechanisms that help to decrease coherence management overhead in directories by not keeping track of portion of data blocks which are private to each core. However, we have observed that performing data classification in a finer granularity than page granularity can bring additional benefits. Based on this observation, this paper makes the following contributions.

- First, we propose a data block classification mechanism which works at subpage level and helps to detect considerably more private data blocks. Consequently, it reduces the percentage of blocks required to be tracked in the directory significantly compared to similar page level classification approaches. This, in turn, enables smaller directory caches with lower associativity to be used in CMPs without hurting performance, thereby helping the directory structure to scale gracefully with the increasing number of cores.

- Second, we propose a small distributed table referred to as the on-chip page table, which stores the page table entries for recently accessed pages in the system. This can be implemented as a portion of the directory controller. Upon a TLB miss, the operating system gets involved in address translation only when the translation is not found in the on-chip page table. It also helps to negate performance degradation that might have occurred due to the increase in the frequency of the operating system's involvement in our subpage granularity block classification.

The rest of the paper is organized as follows. Section 2 discusses the background, related work and motivation for our proposal. In Section 3, we give the details of our approach along with examples, while Section 4 discusses the details of our system configuration, tools, system parameters, and benchmarks. Section 5 gives our experimental results in detail. Finally, Section 6 concludes the paper.

## 2. BACKGROUND AND MOTIVATION

### 2.1 Directory Caches

Directory-based cache-coherence protocols [2, 11] are the common approaches for managing the coherency in many-core systems because of their scalability in power consumption and area compared to traditional broadcast-based protocols. However, the latency and power requirements of today's many-core architectures with their large last level caches (LLCs) brought new challenges. It is common to cache a subset of directory entries to avoid high latency and power overheads of directory accesses. All these motivate architects to cache a subset of directory entries. A directory cache [11] should provide an efficient way to keep the copies of data blocks stored in different private caches coherent since its structure can have momentous influence on overall system performance. The first important decision associated with directory cache design is to decide which data blocks must be tracked in directory cache. Second will be the size and associativity requirement for an efficient directory cache.

A common scheme for organizing directories in CMPs is duplicate-tag-based directories [3, 16]. Compared to other directory structures, this type of directory caches are more flexible as they do not force any inclusion among the cache levels. However, directories based on duplicate-tag come with overheads. Storage cost for duplicate tags, and more notably, high associativity requirement that grows with number of cores in the system, are two main overheads. For instance, in a many-core processor with N cores, each of which has a K-way-set associative last level private cache, the directory cache must be N*K-way associative to hold all private cache tags (to avoid any invalidation). Therefore, this approach suffers from high power consumption and high design complexity due to high associativity.

In our proposed framework, we avoid poor performance of low associative directory caches (as a result of high invalidation counts in directory cache), as will be presented in detail in Section 3.

### 2.2 Fast Address Translation

TLBs are the key component for supporting fast translation from virtual to physical addresses. However, in many cases, they are in the critical path of memory accesses. This is the main motivation for many studies [6, 17] which focus on the techniques for fast and efficient virtual to physical address translation through TLBs.

A pervasive approach for organizing the TLBs in many-core architectures is per-core private TLBs. However, it is shown in [5] that such an organization may lead to poor performance and still keep the TLB in critical path of memory accesses. To overcome this, different techniques were suggested, which are broadly classified into two categories. The approaches in the first category suggest a shared last level TLB [5, 6] to improve sharing capacity among the cores in the system. In contrast, the approaches in the second category try to increase sharing capabilities between the cores by enabling a cooperation between the private TLBs [17]. In other words, each individual core tries to borrow capacity from private TLBs of other cores in the system before finding the translation with relatively higher cost in OS.

The approaches using a shared TLB introduce higher access latency and require a high bandwidth interconnection network. Hence, they need higher associativity, leading to higher power consumption. On the other hand, the approaches in the second category need to inquire other TLBs for each missed page translation happened at each core, which can introduce both design complexity and higher network traffic. For example, a prior study [17] suggests snooping the other TLBs in the system for finding the page entry in those TLBs. This approach is not scalable with higher core counts due to traffic overhead and excessive energy consumption required by snooping.

This motivates us to propose a new method to boost virtual address translation that doesn't suffer from the shortcomings listed above. We present the details of our approach in Section 3.

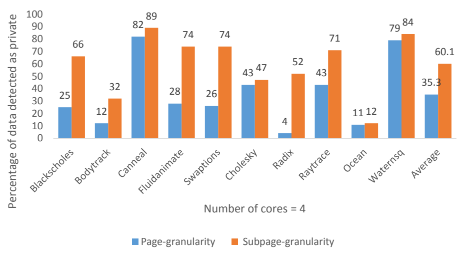
### 2.3 Key Observation

Some prior studies exploited private data detection to enable high performance for many-core architecture by mitigating the overhead of managing coherence. The detection of private data might be done offline with compiler assistance [15]. Although, this approach does not incur any runtime overhead or any extra hardware, however, there is a limitation on the amount of the private data which can be statically detected. Alternatively, detection can be implemented with a runtime technique to increase the private data detection rate [10, 13]. While those approaches show impressive performance gain, the following observation indicates that there is still more room for improvement via detection of private accesses.

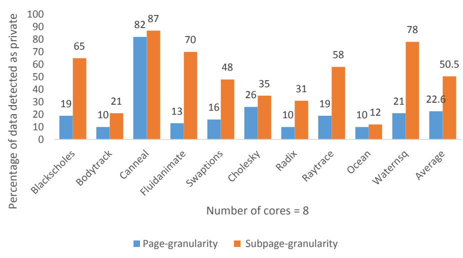
In this work, we also utilize a runtime mechanism to detect private data and further improve the performance of cache coherence management in the system. More specifically, we build our cache coherence management based on the following observation.

**Observation** *The granularity at which we detect private accesses can play a vital role in performance benefits we can obtain.*

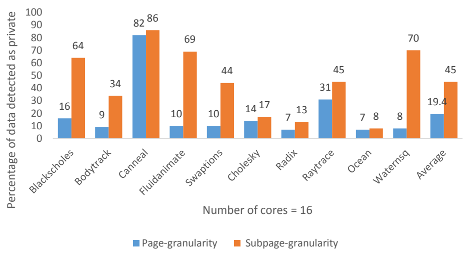
We observed that by inspecting private data in a finer granularity than page granularity, chances for finding private data and further improvements will be considerably higher. Figure 1 shows the amount of private accesses detected with a subpage granularity (4 subpages per page) compared to a page granularity approach for ten different multithreaded applications in systems with 4, 8, and 16 cores. The reason for such a difference is that the existence of a single shared block within a page is enough to change the status of the whole blocks within that page from private to shared. Thus, if we divide a page to subpages, it is more unlikely that shared blocks can adversely affect private blocks' status in



(a) 4 core



(b) 8 core



(c) 16 core

Figure 1: Percentage of data detected as private in a page granularity detection versus subpage granularity (4 subpages per page) detection mechanisms.

subpage level detection.

While the page size is selected as 8KB in our example, we predict the difference would be more dramatic for those architectures that use larger page sizes for performance reasons. According to Figure 1, the chances of detecting private accesses in subpage granularity is about two times more than doing so in page granularity. We will discuss later in this paper how we can perform the detection at the subpage level with the assistance of page tables and TLB entries.

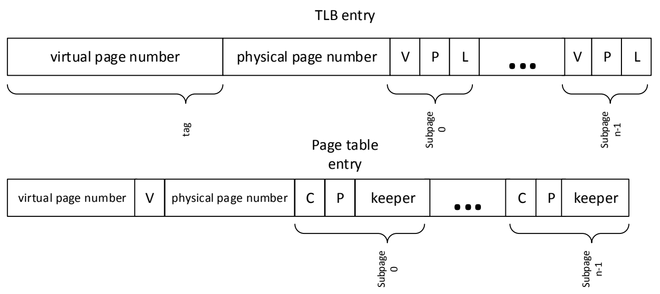
## 3. OUR APPROACH

In this section, we explain the details of our memory management scheme that employs a runtime subpage granularity private data detection motivated by the observation in Section 2. The two main mechanisms used in our approach are i) detecting private memory blocks in subpage granularity and ii) exploiting the results of the data classification to improve the performance of the system.

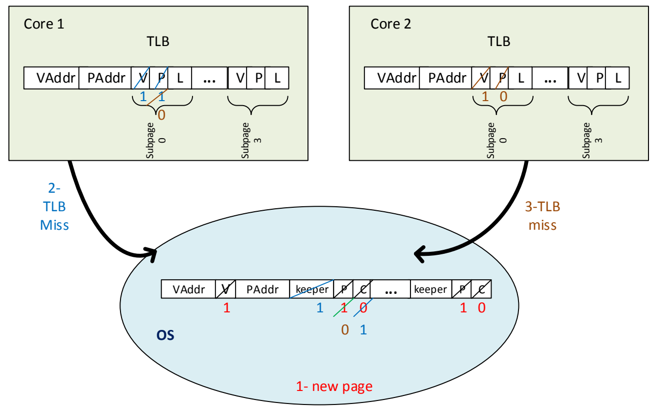
### 3.1 Private Block Detection

A common approach to differentiate between private and shared data blocks is to utilize OS capabilities [10, 12, 13]. The prior work [10] extends TLB and page table entries with additional fields to distinguish between private and shared pages. To do so, two new fields are introduced in TLB entries: while the private bit (P) indicates whether the page is private or shared, the locked bit (L) is employed to prevent race conditions when a private page becomes shared and, in turn, the coherence status of cache blocks of this particular page are restored. To distinguish between private and shared pages, three new fields are also attached to page table entries: (1) the private bit (P) marks whether the page is private or shared; (2) if P is set, the keeper field indicates the identity of the unique core storing the page table entry in its TLB; (3) cached-in-TLB bit (C) shows whether the keeper field is valid or not.

While we also try to detect private data blocks at runtime, our intention is to detect private data at a finer granularity. To accomplish this, we use most significant bits of page offset for subpage ID and clone V, P, C, L and keeper fields in TLB and page table entries so that each subpage has its own such fields, as depicted in Figure 2a. In this work, we divide each page into a number of subpages. The size of the keeper field grows according to the number of cores in the system. In other words, the size of the keeper is $\log_2(N)$, where N is number of cores in the system. Note that extra storage required in page table entries are part of OS storage and does not force any additional storage requirement to



(a) TLB and page table entry formats.



(b) The subpage granularity private data detection mechanism.

Figure 2: Private block detection scheme.

the underlying hardware except the three bits required to identify the status of each subpage in TLB entry.

Now, we list the three main operations that should be performed to properly update the fields discussed earlier and enable detection of private data at subpage level. To make it clear, we also show the operations in Figure 2b with different colors.

- First (red): When a page is loaded into main memory for the first time, the operating system allocates a new page table entry with the virtual to physical address translation. Besides storing the virtual to physical address translation in the page table entry, all the subpages within that page are considered to be private and thereby, the corresponding (P) bits are set. All subpages' bits (C) are also cleared, showing that the entry has not been cached in any TLB yet.

- Second (blue): Core1 faces a miss in its TLB for an address translation or there is a hit in the TLB but the (V) bit of the subpage which was tried to be accessed, is cleared (which means it is not cached in TLB yet). In either case, core1 will inquire the operating system page table for the translation of the subpage. We assume that it finds the (C) bit of the subpage cleared, which means that the subpage is not accessed by any other core yet. Thus, the (C) bit is set and the identity of the requester core (core 1) is recorded in the keeper field.

- Third (brown): Core2 experiences a miss in its TLB for the same subpage like the previous operation. After looking up the page table for that subpage, it turns out both (C) and (P) bits of the subpage are set. Therefore, the keeper field should be compared against the identity of the requester core. If there is a match, it means that the keeper core has already experienced a TLB miss and the page table entry is brought into the requester core's TLB, considering the subpage as privately accessed only by requester core. If the keeper field does not match the identity of the core requesting the page table entry (like this example), it means that two different cores are attempting to access the data within the same subpage (core1 and core2 in this example). As a result, the operating system decides to turn the status of corresponding subpage to shared by clearing the (P) bit. Moreover, the operating system triggers the coherence recovery mechanism by informing the keeper core to restore the coherence status of cache blocks within that subpage. We will explain the coherence recovery mechanism with more detail in the next part.

#### 3.1.1 Coherence Recovery Mechanism

As will be discussed, the performance improvement we expect to gain by our approach is based on the fact that one can ***avoid*** coherence operations for ***private*** data. In fact, private data does not necessarily need many of the messages exchanged between the cache controllers. Similarly, the directory cache does not need to track private blocks.

If at a certain point, we realize that our assumption about the status of a block is no longer valid, we need to recover from this situation. Otherwise, the caches might not remain coherent. In this work, we use a similar recovery mechanism proposed in the literature [10]. In this work, authors propose two strategies, namely, flushing-based recovery and updating-based recovery mechanisms. Their results show that these two strategies are slightly different in terms of performance. Similarly, our recovery mechanism uses a flushing-based mechanism and performs following operations in order to ensure safe recovery from status change of a subpage from private to shared.

- First, on the arrival of recovery request, the keeper first should prevent accesses to the blocks of that subpage by setting the subpage's (L) bit.

- Second, the keeper should invalidate all the blocks corresponding to that subpage in its private cache.

- Third, the keeper also should take care of the pending blocks in its Miss Status Holding Register (MSHR). If there are any blocks within that subpage in MSHR, they should be evicted right after the operation completes.

Once the mentioned operations finish, the keeper sends back an acknowledgment to announce the completion of the recovery. At this point, the core which initiated the recovery, changes the subpage status of that specific subpage to shared and continues its operation.

## 3.2 Directory Cache Organization

In this section, we present our approach which tries to address the two most important drawbacks of directory-based cache coherence protocols. The first drawback is the need for a highly associative cache, which introduces high power consumption and high complexity in the design. Second, the high storage cost for keeping track of all the blocks exist in the last level private caches. Unfortunately, both of these drawbacks threaten the scalability of the system. Moreover, the former linearly gets worse with the number of cores in the system as depicted in Section 2.

The primary solution for solving the discussed scalability problem inherited in the duplicate-tag directory based cache coherence protocols is adjusting the associativity value of directory cache to some low values similar to the ones in the associativity of private caches. However, with this approach, the number of evictions in directory caches caused by adding a new entry to the directory cache, might increase dramatically. Since any eviction in directory cache requires invalidation of all the copies of the corresponding cache block (in all the private caches in the system); performance of the system will be jeopardized. Thus, the directory cache has the potential to become a major bottleneck for large-scale many-core architectures.

In this work, we try to utilize our private data detection to address the aforementioned scalability problem as follows. As we discussed earlier, we do not need to keep track of the private data. Thus, we can avoid polluting the directory cache with the private data. We simply hold the states for the shared data and not for the private ones. As we will show in the evaluation section, we can dramatically decrease the directory cache eviction rate and mitigate the inevitable performance degradation due to the high directory eviction counts. The invalidation of the blocks related to the evicted directory entries is performed as normal. In the sensitivity analysis, we show that our idea can work with different associativity values and we can still get acceptable performance results even with the low associativity.

## 3.3 On-chip Page Table

Memory block classification at subpage level may increase the frequency of the operating system's involvement in updating the maintenance bits belonging to subpages stored in page table entries, nullifying some portion of performance benefits of subpage level data classification. For this reason, as our second contribution, we show how we can negate the possible performance degradation by introducing on-chip page table. Moreover, the proposed method also enables us to boost the performance of the virtual memory management in many-core systems.

Based on the drawbacks of previous studies on TLB organization (see section 2), we propose our on-chip page table as follows.

1- We increase the probability of accessing a page translation within the chip by introducing excessive capacity rather than the private TLBs for keeping the page translation on the chip. In our system, each core's private TLB includes 64 entries, thereby using the same size on-chip page table per core. We reserve this space for our *on-chip page table* by not keeping the private data status in the directory cache. In the experimental results, we show how devoting even a small portion of the directory cache to the page translation can make the translation faster without adversely affecting the performance of other components.

2- In contrast to the discussed approaches, we avoid snooping by distributing the pages based on their tags into the on-chip page tables located in the directory controller. Distribution is done by interleaving the page entries according to the least significant bits of virtual page numbers (for example, with 16 cores, we use 4 least significant bits). This way, for each miss in the private TLBs, the core sends the request for the page address translation only to one of the cores' on-chip page table. This makes our method more scalable compared to other proposed cooperative TLBs based on snooping.

Figure 3 depicts the structure of our on-chip page table and how a miss in one of the private TLBs can be resolved by one of the on-chip page tables. After a TLB miss occurs in core 0 for an address translation of page 'a', the request for finding the page information for page 'a' is sent directly to the core which may have the entries for that page (in this example core N-1). Then, the translation is forwarded back to the requested core in case it is found in the on-chip page table (red line). In the second example, the search for finding the translation for page 'b' was not successfully found in the corresponding on-chip page table. Therefore, with OS involvement, the entries will be written to one of the on-chip page tables after interleaving (core1) and also the TLB of the requested core (core0).

The proposed approach does not force major hardware costs nor operation overheads. For each TLB miss, there is an *address interleaving* (which can easily be done by a shift and AND operation) to find the location of the on-chip page table that might have the corresponding physical address for a virtual address. As on-chip page tables reside in the directory caches, each access to an on-chip page table is equivalent to an access to a directory cache. This implies that we can replace a very costly page table walk with a very low cost cache access, each time the access to the on-chip page table is a hit.

In our experiments, we show how much we can avoid referring to OS, thanks to exploiting the on-chip page table. To show increasing the capacity of the TLB can not solely decrease TLB miss ratio, we also compare our method with the system with different TLB sizes.

## 4. EXPERIMENTAL SETUP

### 4.1 System Setup

We evaluate our proposal with gem5 full-system simulator [8] running linux version 2.6. gem5 uses RUBY, which implements a detailed model for the memory subsystem and specifically cache coherence protocol. For modeling the interconnection network, we use GARNET [1], a detailed interconnection simulator also included in gem5. We apply

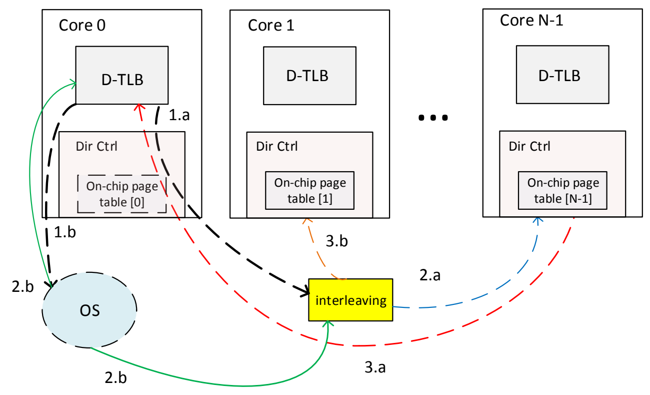


Figure 3: The structure of on-chip page table in a CMP with private per-core TLBs.

our idea to *MOESI-CMP-Directory* which is a directory-based cache coherence protocol implemented in gem5. We present results with a system consisting of 16 cores with level one private data and instruction caches, and a shared level two cache. Table 1a provides details of our simulation environment. In the rest of the paper, this configuration is considered as the base setup, where our private/shared data classification approach is not applied.

### 4.2 Benchmarks

We evaluate our proposal with ten different parallel workloads from two commonly used suites (SPLASH-2 [18] and PARSEC 2.1 [7]). As we execute a large set of simulations to provide a comprehensive sensitivity analysis, where each simulation takes a considerable amount of time to run in our full-system simulator, thus, we simulated the applications mostly for small size data-sets, as indicated in Table 1b.

For our experimental results, we only consider the parallel phase of benchmarks or Region of Interest (ROI); and the number of threads used by each application is set based on the number of cores in the system, which is 16 by default.

### 4.3 Evaluation Metrics

We compare our approach with a state-of-the-art directory-based cache coherence protocol. The specific metrics we tested are the number of evictions in the directory cache, miss ratio, the network traffic, and the execution time.

In a directory cache based on duplicate-tag, the associativity of directory cache and its size are the limiting factors for scalability. We test the scalability of our approach against a state-of-the-art directory cache coherence protocol. Thus, we measured the performance of our approach with directory cache associativity equal to 4; while in the base setup, the associativity is set to 64.

## 5. EVALUATION

In this section, we show how our proposal is able to improve the performance of directory-based cache coherence protocol through reduction in cache miss ratio, network traffic, and latency of resolving cache misses. Therefore, we first show the reduction in the number of evictions from the directory cache. Then, we show the effects in cache misses as well as number of messages interchanged in the network

| | |
|---|---|
| Processor | 16 Alpha cores, 2GHz, 64 entries TLB |
| Private L1D | 32 KB (=**512** entries), 4-way associative, 64B cache-block size |
| Private L1I | 32 KB, 2-way associative |
| Shared L2 | 32MB (2MB per core) , 8-way associative |
| Directory cache | **512** entries, 64 way associative |
| Network | Torus , Fixed Garnet interconnection model |
| Cache coherence protocol | MOESI CMP Directory |
| Page size | 8KB |

(a) Default simulation parameters used in our experiments.

| Benchmarks | Input |
|---|---|
| **Parsec 2.1** | |
| Blackscholes | simsmall |
| Bodytrack | |
| Canneal | |
| Fluidaniate | |
| Swaptions | |
| **Splash 2** | |
| Cholesky | tk15.O |
| Raytrace | Teapot environment |
| Waternsq | 512 molecules, 3 time steps |
| Radix | 1048576 keys |
| Ocean | 258*258 ocean |

(b) Benchmarks and respective input files.

Table 1: Experimental setup.

as a result of better directory cache eviction rate. We also study the performance of our proposal in reducing the latency of resolving a cache miss. Moreover, we evaluate our on-chip page table by showing the amount of misses in private TLBs that can be resolved by introducing our on-chip page table. Then, we have a discussion on how the number of subpages should be set for the best performance results. Finally, we discuss the effects of using lower associativity directory caches.

## 5.1 Directory Cache Eviction

As we showed earlier, a considerable amount of accessed memory blocks are private; and in our proposal, we do not keep track of those blocks in directory caches. By not polluting the directory cache with status of private blocks that do not need coherence maintenance, we would have less eviction in directory cache even for caches with lower associativity. Figure 4a shows directory cache eviction rate of our proposal normalized to the base configuration. As can be seen, on average, we have 58% less evictions in the directory cache compared to base setup without any data classification.

There are two main factors for our approach to improve the eviction rate of the directory cache. First one is the ability to detect private data blocks, whereas, the second one is the shared block access pattern. Arguably, if we can detect more private data, we can avoid more evictions in the directory caches. For instance, number of directory evictions has been reduced for Waternsq more than Cholesky, since we were able to detect higher number of private data blocks for Waternsq compared to Cholesky (70% for Waternsq and 17% for Cholesky). However, it is not the only factor which affects reduction in the directory eviction. The sharing pattern of an application can also play an important role in the number of evictions that happen in the directory. For instance, for Canneal, we were able to detect a high percentage of private data blocks (86%) compared to Bodytrack (34%). But, because of the difference in sharing pattern for these two applications, we observe rather same normalized eviction rate. The reason is due to the temporal communication behavior of these two applications. In Canneal, the communication between the cores take place throughout the execution of the application, whereas, in Bodytrack, for the majority of the parallel phase, there is limited communication between cores [4]. Therefore, in Canneal, the chance for a possible conflict will be higher than Bodytrack.

Another observation in Figure 4a is the dramatic improvement in directory eviction ratio for Waternsq application. Based on the results reported in [4], in Waternsq and Waterspa (is not used in this work!) *all cores* are actively involved in a producer-consumer pattern. Furthermore, based on this observation, they conclude that a broadcast-based technique is likely to benefit from this in Waternsq and Waterspa. We also observe that, Waternsq benefits the most, considering the eviction rate aggregation for all the directory caches in the system.

## 5.2 Private Cache Misses

One of the primary advantages of reducing directory cache eviction is the reduced invalidations at the last level private caches (in our system, the L1 cache). This is due to the fact that any eviction of a block in directory cache implies invalidation of all the blocks in any of the L1 caches that correspond to that block. Figure 4b shows the L1 cache miss ratio for 10 different multi-threaded applications normalized with respect to the base case. Through our approach, we have about 15% reduction in private L1 cache miss ratio.

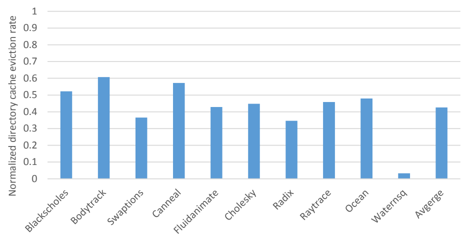
We have better normalized the cache miss values, for those applications with fewer misclassified blocks. In other words, the higher the number of private blocks detected out of all actual private data blocks exist, the higher the reduction is in the number of cache misses.

## 5.3 Network Traffic

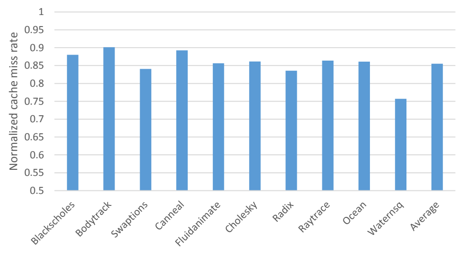
The number of messages exchanged in the system is also reduced by our proposal. This is because evictions in directory caches and processor cache misses impact the network message counts. For instance, for resolving a miss in the first level private cache, different controllers in the system need to exchange request, forward, and response control/data messages with one another. In Figure 4c, we compare the message counts of our proposal normalized with respect to the base system. Our approach reduces the message traffic between 9% and 21% with an average of 12%.

## 5.4 Cache Miss Latency

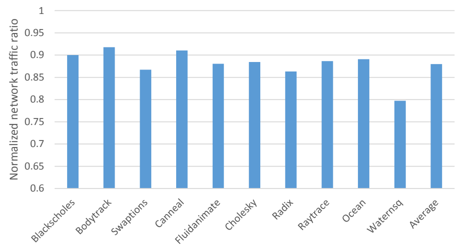
With our proposal, we are also able to reduce the average latency for resolving cache misses. More specifically, we avoid referring to the directory cache for those requests associated to the private blocks. Therefore, some portion of the requests experience less latency with a cache miss lowering the overall average latency of a cache miss. Figure 4d depicts the average cache miss latency normalized to the base case for our applications. As can be seen, on average,



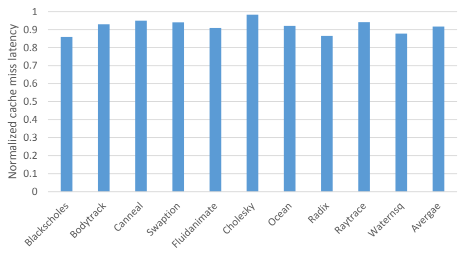
(a) Normalized directory cache eviction rate.



(b) Normalized L1 cache miss rate.



(c) Normalized network traffic message count.



(d) Normalized cache miss latency.

Figure 4: Normalized directory cache eviction rate (a), L1 cache miss rate (b), coherence traffic rate (c) and cache miss latency (d).

we resolve the cache misses 8% faster than the base system.

## 5.5 Performance of Exclusive TLB

In this part we show the improvements of virtual memory management by introducing the on-chip page table. Figure 5 shows the percentage of TLB misses that also experience a miss in on-chip page tables. In other words, it shows how much we can avoid accessing the costly OS page table by finding the required page translation in the distributed page table. As an example, for Canneal benchmark, only 8 % of page translations which causes a miss in private TLBs, can not be found in the on-chip page table. Therefore, the remaining 92% of accesses find the right translation in the on-chip page table after they experienced a miss in private TLBs. On average, our approach prevents 84% of accesses to operating system by introducing the on-chip page table.

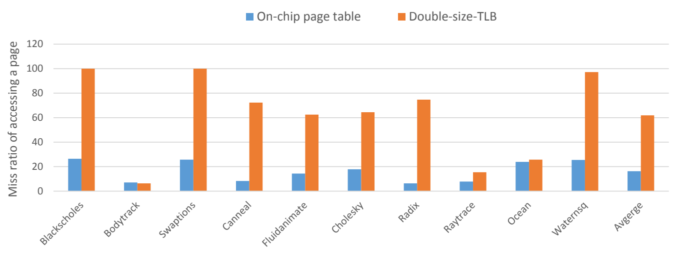


Figure 5: Normalized miss rate of finding page translation in the core.

We also compared our on-chip page table with a modified version of the base system, where the TLB is double the original size. As can be seen in Figure 5, by doubling the TLB size, we can only eliminate the 39% of the accesses to OS page table. So, we can conclude that increasing the caching capacity for page translation can not solely improve the performance. Moreover, exploiting an effective technique which enables sharing page translation between the cores is also very crucial for providing a fast virtual page translation. As it can be seen, in most of the benchmarks, there is a big difference between the miss rate for on-chip page table and double-size TLB which proves the former statement.

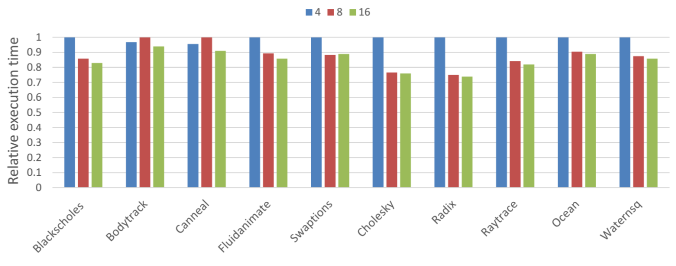
## 5.6 Subpage Size

In this section, we explore how much we can increase granularity of private data detection in our approach to improve performance. For doing so, we run a set of simulations for different subpage sizes. Figure 6a shows relative execution times for three different number of subpages per page (4, 8, and 16). As can be seen, in all the benchmarks except Bodytrack and Canneal, a system with subpage size 8, shows better performance compared to a system with subpage size 4. Canneal and Bodytrack, however, benefits from a system with 4 subpages per page. The other observation is that a system with subpage size 16 shows the best performance for all the benchmarks. Moreover, subpage sizes 8 and 16 show almost similar performance in most of the applications tested.

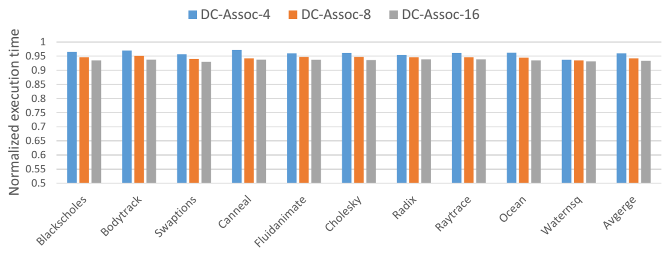
The most important conclusion on subpages is that increasing the granularity does translate into performance improvement up to a certain point. In another words, the performance improvement achieved by detecting more private data is offset by the overheads of detecting private data at a finer granularity.

## 5.7 Execution Time and Sensitivity Analysis

The reduction in the number of cache misses, the reduced latency in resolving them, and a decline in the number of messages exchanged in the network; all result in a reduction in the latency of executing the application as it is shown in Figure 6b. This figure also illustrates the sensitivity of our approach to the associativity of directory cache. The three bars labeled as DS-Assoc-4, DS-Assoc-8, DS-Assoc-16 represent three configurations with directory cache associativities equal to 4, 8, and 16, respectively. On average, we generate 4%, 6%, and 7% improvement in execution time for directory caches with associativities equal to 4, 8, and 16, respectively. One key observation based on the results in Figure 6b is that our approach enables quite similar results even for directory caches with lower associativity.



(a) Relative execution time for different subpage sizes. Blue, red and green bars shows the relative execution time for 4, 8 and 16 subpage sizes respectively.



(b) Execution time normalized with respect to the base system. DC-Assoc-4, DC-Assoc-8, and DC-Assoc-16 stand for directory caches with associativity equal to 4, 8 and 16, respectively.

Figure 6: Sensitivity results.

## 6. CONCLUSION

In this paper, we propose a subpage granularity data management scheme which improves the performance of directory-based cache coherence protocols by decreasing the number of evictions taking place in directory caches. This is done by applying a subpage granularity data classification which helps us not to keep track of significant percentage of data blocks in directory caches. We observed that performance improvement stops after a certain granularity level. Moreover, we also accelerate virtual to physical address translation by introducing on-chip page table. Specifically, we avoid 84% of accesses to the OS page table. Overall, we observe up to 7% improvement in execution time even for directory caches with lower associativities, which ensures the scalability of our approach.

## 7. ACKNOWLEDGMENT

This study was fully funded by the Scientific and Technological Research Council of Turkey (TUBITAK) with a grant 113E258.

## 8. REFERENCES

[1] N. Agarwal, T. Krishna, L.-S. Peh, and N. K. Jha. GARNET: A detailed on-chip network model inside a full-system simulator. In IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pages 33-42, April 2009.

[2] A. Agarwal, R. Simoni, J. L. Hennessy, M. A. Horowitz. An evaluation of directory schemes for cache coherence. In 15th International Symposium on Computer Architecture (ISCA), pages 280-289, May 1988.

[3] L. A. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S.Qadeer, B.Sano, S.Smith, R.Stets, B.Verghese. Piranha. a scalable architecture based on single-chip multiprocessing. In 27th International Symposium on Computer Architecture (ISCA), pages 12-14, June 2000.

[4] N. Barrow-Williams, C. Fensch, S. Moore. A communication characterisation of Splash-2 and Parsec. IEEE International Symposium on Workload Characterization (IISWC), pages 86-97, October 2009.

[5] A. Bhattacharjee, D. Lustig, M. Martonosi. Shared last-level TLBs for chip multiprocessors, In 17th Symposium on High Performance Computer Architecture (HPCA), pages 62-63, February 2011.

[6] A. Bhattacharjee and M. Martonosi. Characterizing the TLB behavior of emerging parallel workloads on chip multiprocessors. In International Conference on Parallel Architectures and Compilation Techniques (PACT), pages 29-10, September 2009.

[7] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC benchmark suite: Characterization and architectural implications. In 17th International Conference on Parallel Architectures and Compilation Techniques (PACT), pages 72-81, October 2008.

[8] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. The gem5 simulator, SIGARCH Comput. Archit. News, vol 39, no 2, pages 1-7, 2011.

[9] P. Conway, N. Kalyanasundharam, G. Donley, K. Lepak, and B. Hughes. Cache hierarchy and memory subsystem of the AMD Opteron Processor. IEEE Micro, pages 16-29, March/April 2010.

[10] B. Cuesta, A. Ros, M. E. Gomez, A. Robles, and J. Duato. Increasing the effectiveness of directory caches by deactivating coherence for private memory blocks. In 11th International Symposium on Computer Architecture (ISCA), pages 184-195, June 2011.

[11] A. Gupta, W. Weber, and T. Mowry. Reducing memory and traffic requirements for scalable directory-based cache coherence schemes. In Proceedings of the International Conference on Parallel Processing, 1990.

[12] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki. Reactive NUCA: Near-optimal block placement and replication in distributed caches. In 36th Int'l Symp. on Computer Architecture (ISCA), pages 184-195, June 2009.

[13] D. Kim, J. Ahn, J. Kim, and J. Huh. Subspace snooping: Filtering snoops with operating system support. In 19th Int'l Conference on Parallel Architectures and Compilation Techniques (PACT), pages 111-122, Sept. 2010.

[14] M. R. Marty and M. D. Hill. Virtual hierarchies to support server consolidation. In Proceedings of the 34th Annual International Symposium on Computer Architecture, June 2007.

[15] Y. Li, R. Melhem, A.K. Jones. A practical data classification framework for scalable and high performance chip-multiprocessors. IEEE Transactions on Computers, vol.63, no.12, pages 2905-2918, December 2014.

[16] A. K. Nanda, A. Nguyen, M. M. Michael, D. J. Joseph. High-throughput coherence control and hardware messaging in Everest. IBM Journal of Research and Development 45 (2) (2001), pages 229-244.

[17] S. Srikantaiah, M. Kandemir. Synergistic TLBs for high performance address translation in chip multiprocessors. In 43rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pages 313-324, December 2010.

[18] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, A. Gupta. The SPLASH-2 programs: Characterization and methodological considerations. In 22nd International Symposium on Computer Architecture (ISCA), pages 24-36, June 1995.