Int J Parallel Prog (2014) 42:435–455
DOI 10.1007/s10766-013-0247-8

Bandwidth Adaptive Cache Coherence Optimizations for Chip Multiprocessors

Abdullah Kayi · Olivier Serres · Tarek El-Ghazawi

Received: 4 November 2012 / Accepted: 13 April 2013 / Published online: 1 May 2013

Abstract Chip Multiprocessors (CMPs) have different technological parameters and physical constraints than earlier multi-processor systems, which should be taken into consideration when designing cache coherence protocols. Also, contemporary cache coherence protocols use invalidate schemes that are known to generate a high number of coherence misses. This is especially true under producer-consumer sharing patterns that can become a performance bottleneck as the number of cores increases. This paper presents two mechanisms to design efficient and scalable cache coherence protocols for CMPs. First, we propose an adaptive hybrid protocol to reduce coherence misses observed in write-invalidate based protocols. The proposed protocol is based on a write-invalidate scheme. However, adaptively, it can push updates to potential consumers based on observed producer-consumer sharing patterns. Secondly, we extend this adaptive protocol with an interconnection resource aware mechanism. Experimental evaluations, conducted on a tiled-CMP via full-system simulation, were used to assess the performance from our proposed dynamic hybrid protocols. Performance analysis is presented on a set of scientific applications from the SPLASH-2 and NAS parallel benchmark suites. Results showed that the proposed mechanisms reduce cache-to-cache sharing misses up to 48 % and speed up application performance up to 34 %. In addition, the proposed interconnection resource aware mechanism is proven to perform well under varying interconnection utilizations.

A. Kayi (✉)
Intel PTD, Hillsboro, OR, USA
e-mail: abdullah.kayi@intel.com

O. Serres · T. El-Ghazawi
The George Washington University, Washington, DC, USA
e-mail: serres@gwu.edu

T. El-Ghazawi
e-mail: tarek@gwu.edu

Keywords Cache coherence · Multi-core · Bandwidth adaptive

1 Introduction

The density of transistors inside a single chip has continued to increase exponentially as stated in Moore's law [30]. However due to power dissipation, thermal constraints and limited instruction-level parallelism (ILP), major microprocessor vendors have shifted their designs into utilizing the available number of transistors inside a single chip by using multiple homogeneous cores resulting in multi-core architectures or also known as Chip Multiprocessors (CMPs) [4,15]. CMPs provide a solution to increase the performance capability on a single chip without requiring a complex system which increases power requirements [7,17]. Hence, CMPs have become the *de facto* architecture choice for processor design.

This trend will likely continue with more and more cores being put on a single chip [17]. This transition to multi-core processors provides new ways of reaching performance gains mainly through Thread-Level Parallelism (TLP). TLP requires parallel programming, and accordingly most CMPs have adapted the shared-memory model for ease of use. Albeit the ease of use in programmability, shared memory view brings the overhead of memory consistency and coherency problems. Recent research has demonstrated the implications of the cache coherence overhead on application performance including large high-performance clusters with multi-socket CMP based shared memory nodes [22]. With the increasing number of cores inside a single chip, cache coherence mechanism becomes even more important for performance growth in CMP based systems.

Although cache coherence protocols were extensively studied for earlier shared-memory systems, with the advent of CMPs cache coherence protocol has become a first-order design issue at the chip level. In addition, CMPs have different technological parameters and physical constraints which should be considered for cache coherence protocol design to achieve better performance and scalability. On-chip resource utilization is crucial for CMPs, a remote but on-chip cache-to-cache access is less costly than off-chip memory access which is in contrast to the earlier multiprocessors [13]. This calls for a cache coherence protocol design which puts more emphasis on avoiding off-chip accesses as much as possible.

Thereby, cache coherence overhead can still be amortized while facilitating many-cores on a single chip. In accordance with these technological changes, this paper proposes cache coherence mechanisms optimized by taking into account the aforementioned CMP specific parameters and constraints.

There are two broad sets of cache coherence protocols based on the underlying mechanism to satisfy coherency: broadcast-based snooping protocols and directory-based protocols. Snooping protocols are known to have scalability issues due to high bandwidth usage and also, most of the time, they rely on ordered interconnects [28]. Therefore, our proposed cache coherence protocol is based on a directory scheme for better scalability. Cache coherence protocols can also be divided into two main categories based on their write policies on shared data: *write-invalidate* (WI) and *write-update* (WU) protocols. In a write-invalidate protocol, the processor that is about to write to a shared data block first invalidates the other copies of the same block and then can update its own local copy. Having the other copies invalidated, further modifications to that block does not require global traffic between processors. The advantage of the write-invalidate strategy is the fact that subsequent write operations can be completed locally until the same data block is accessed by another processor. In contrast, write-update protocols maintain cache coherency by sending the updates to all other sharing processors for each and every modification. This eliminates cache-to-cache misses for those data blocks that observe producer-consumer sharing pattern. Producer-consumer sharing pattern is defined in which one processor modifies a data block (producer) and a set of other processors read the block (consumers) [6]. However, write-update strategy results in extra traffic especially for those cases when the updated block will not be used by the receiving processors before the modifying processor is done with its updates (e.g. migratory-sharing data) [5,8,36]. On the other hand, write-invalidate based protocols tend to generate high cache-to-cache misses. Besides, previous studies have illustrated that these cache-to-cache misses comprise a big portion of overall cache misses in shared memory systems [2,12,29] and write-update protocols can degrade these coherence misses [6,18,35]. Hence, there is no single cache coherence solution that can perform well under varying workload characteristics, different cache configurations, and interconnect parameters [28]. As a solution, this work proposes an adaptive, hybrid protocol which switches between write-updates and write-invalidates based on the underlying producer-consumer sharing patterns. For clarity, this protocol will be referred as Producer-Consumer Adaptive (PCA). This approach requires only a minimalistic hardware extension to benefit from such adaptive protocols. Also, the proposed mechanisms do not change the memory model seen by the programmer or by the compiler. Proposed adaptive protocol can be implemented on top of any write-invalidate protocol and can work with existing language, compiler, and processor designs. Since most of the contemporary systems use write-invalidate based protocols, this paper provides an adaptive protocol based on a write-invalidate directory protocol that triggers write-updates based on producer-consumer sharing pattern detection. In addition, PCA is designed and optimized considering CMP specific topological constraints in which on-chip communication is prioritized compared to off-chip memory accesses in order to satisfy any data request.

Cache coherence protocols can be divided into two main categories based on their write policies on shared data: *write-invalidate* (WI) and *write-update* (WU) protocols. In a write-invalidate protocol, the processor that is about to write to a shared data block first invalidates the other copies of the same block and then can update its own local copy. In contrast, write-update protocols maintain cache coherency by sending the updates to all other sharing processors for each and every modification. This eliminates cache-to-cache misses for those data blocks that observe producer-consumer sharing pattern. Producer-consumer sharing pattern is defined in which one processor modifies a data block (producer) and a set of other processors read the block (consumers) [6]. However, write-update strategy results in extra traffic especially for those cases when the updated block will not be used by the receiving processors before the modifying processor is done with its updates (e.g. migratory-sharing data) [8]. On the other hand, write-invalidate based protocols tend to generate a high number of cache-to-cache misses. Besides, previous studies have illustrated that these cache-to-cache misses comprise a big portion of the overall cache misses in shared memory systems [2,12,29] and write-update protocols can degrade these coherence misses [18,35]. Hence, there is no single cache coherence solution that can perform well under varying workload characteristics, different cache configurations, and interconnect parameters [28]. As a solution, this work proposes an adaptive, hybrid protocol which switches between write-updates and write-invalidates based on the underlying producer-consumer sharing patterns. For clarity, this protocol will be referred to as PCA. This approach requires only a minimalistic hardware extension. Also, the proposed mechanisms do not change the memory model seen by the programmer or by the compiler. The proposed adaptive protocol can be implemented on top of any write-invalidate protocol. Since most of the contemporary systems use write-invalidate based protocols, this paper proposes an adaptive protocol based on a write-invalidate directory protocol that triggers write-updates based on producer-consumer sharing pattern detection. In addition, PCA is designed and optimized considering CMP specific topological constraints in which on-chip communication is prioritized compared to off-chip memory accesses.

PCA generates less network traffic compared to a typical write-update protocol as updates only happen for established producer-consumer sharing data blocks to prevent coherence misses. The rest of the blocks are kept coherent via a write-invalidate scheme which, in nature, entails less bandwidth. However, PCA might still yield to hot spots are still possible for applications observing large amount of producer-consumer sharing data blocks during some execution phases. Therefore, we also propose a bandwidth-adaptive scheme to further extend our adaptive, hybrid cache coherence protocol to prevent possible network congestions and the associated performance losses due to updates. This protocol will be referred to as Producer-Consumer Bandwidth-Adaptive (PCBA).

The PCBA protocol utilizes a dynamic feedback control mechanism to monitor and estimate the underlying interconnect utilization. Based on the recent bandwidth utilization information, it makes intelligent decisions whether to trigger the update optimizations or not. As such, PCBA can eliminate potential hot spots and on-chip interconnect congestion due to write-update policy on producer-consumer sharing data blocks.

To summarize, this paper makes the following contributions:

– An adaptive, hybrid directory-based cache coherence protocol that switches from a write-invalidate protocol to a write-update protocol for producer-consumer sharing data blocks to reduce coherence misses. Our work mainly differs from previous work in the CMP specific optimizations that observes the importance of on-chip communication for performance. Also, our proposed protocols have a modest hardware cost which is desirable for CMPs. More information on how our work differs from the other adaptive, hybrid protocol studies can be found in Sects. 2 and 4.3.
– To the best of our knowledge, this is the first cache coherence protocol that utilizes a bandwidth-adaptive mechanism to implement a hybrid protocol that switches between write-invalidate and write-update policies.
– A novel trigger update mechanism to push updates to potential consumers (Sect. 4.4).

– Experimental evaluation of proposed protocols under various workloads and configurations. This includes 16- and 256-core tiled CMP based results.

The rest of this paper is organized as follows: Sect. 2 gives an overview of the related work; Sect. 3 discusses some background information on the experimented tiled-CMP architecture and the base directory cache coherence protocol. Sect. 4 describes the implementation details for the proposed cache coherence protocols along with the producer consumer sharing pattern detection and bandwidth adaptive mechanisms. Also, this section covers the verification of the adaptive protocol. The simulator environment as well as the simulation workloads are explained in Sect. 5. Also, Sect. 5 presents and analyses the experimental results; and Sect. 6 concludes the study and highlights future work.

2 Related Work

Although cache coherence protocols have been the main focus of many earlier research efforts, the burst of CMPs calls for a cache coherence re-design to satisfy the CMP specific technological constraints. These previous studies targeted Multi-Processor (MP) systems comprised of multiple separate single-core processors. The topological constraints differ in MP and CMP designs. Thus, in this paper we propose cache coherence mechanisms that are designed and optimized considering CMP specific constraints.

There have been various research efforts on adaptive and hybrid cache coherence protocols. Write-invalidate versus write-update protocol design has been investigated in many earlier studies [5,6,8–11,19,32,35,36]. All of these studies except [6,18,32] utilized bus interconnect to establish a total order required by cache coherence protocols. Our work is based on a directory protocol and does not require an ordered interconnect to satisfy coherency. We believe that future CMP based systems will have to use a directory like structure to provide coherence and scale to large on-chip core numbers. On the other hand, Nilsson et al. [32] and Grahn et al. [18] studied the performance of a directory-based hybrid protocol, called *competitive-update*, that uses a relaxed memory model to overlap write latency with local processing. Performance study showed lower miss rates for all applications tested, but some applications could not translate this into a performance improvement in the execution time because of an increased network traffic under *competitive update*. Accordingly, they proposed to extend the competitive-update protocol with a migratory data detection mechanism to reduce the network traffic. Our proposed protocol triggers write-updates only for producer-consumer sharing blocks as also studied in [6]. Cheng et al. [6] studied such a sharing pattern and corresponding prediction mechanisms in the context of cache-coherent Non-Uniform Memory Access (ccNUMA) shared memory multiprocessors. This proposed cache coherence protocol entails off-chip memory access for each write-update which is a big performance bottleneck in CMPs. Hence, we develop a cache coherence protocol that can trigger updates to potential consumers without accessing off-chip memory and avoid unnecessary invalidations. Furthermore, to control network traffic we propose a novel bandwidth-adaptive cache coherence mechanism that dynamically shuts down the updates under limited available bandwidth. As such, not only we propose an adaptive, hybrid protocol optimized for producer-consumer sharing pattern to reduce coherence misses but also an enhanced cache coherence protocol with bandwidth awareness to avoid performance losses due to network congestion. In addition, our work does not require Remote Access Cache (RAC) to push the updates which lowers the hardware cost significantly compared to [6].

Bandwidth-adaptive cache coherence protocol was studied earlier by Martin et al. [28] to dynamically adapt to the available bandwidth. However, their work utilized such bandwidth mechanism to switch between a broadcast-based protocol to a directory like protocol. We utilize the bandwidth-adaptive framework to help our hybrid/adaptive protocol to intelligently trigger write-update optimizations based on the available bandwidth. This mechanism is proven to be crucial under limited bandwidth to avoid possible network congestions and corresponding performance losses.

Another approach to deal with cache coherence overheads is based on coherence predictors. Mukherjee and Hill [31] proposed the Cosmos coherence message predictor with an extended study on Yeh and Patt's two-level PAp branch predictor [38]. Kaxiras et al. [20,21] proposed instruction-based predictors as an alternative to address-based predictors to move the shared data close to the consumers as soon as possible. Lai et al. [23] proposed pattern-based memory sharing predictors which reduces the memory requirements of the previous proposed predictors. Acacio et al. [2] devised a prediction scheme for owner prediction to convert 3-hop cache-to-cache misses to 2-hop misses and they utilized prediction schemes to tackle upgrade misses in [3]. Coherence predictor cache was designed and evaluated in [33] as a resource-efficient coherence predictor. Nilsson et al. [33] made an observation based on SPLASH-2 kernels that coherence activity footprint is confined to a small fraction of the whole data set. Further, they utilized address filtering to avoid unnecessary caching to the Coherence Predictor Cache. Martin et al. [27] analyzed commercial workloads and according to the sharing patterns of the observed workloads they proposed destination-set prediction to reduce indirections caused by directory protocols. Perceptron-based coherence predictors were suggested and experimented within [16,24]. Most of these extensions require major modifications and are relatively costly for a CMP design. Our prediction mechanism has relatively a modest overhead with only 19 KB per core as described in Sect. 4.

3 Baseline Architecture

This section provides details about the architecture used and the baseline write-invalidate cache coherence protocol.

3.1 Tiled-CMP Architecture

Tiled-CMP architectures provide a scalable and efficient solution for CMP designs [14]. Accordingly for the experiments, we utilized a tiled-CMP with 16 cores designed via 4×4 2D-MESH interconnect topology as can be seen in Fig. 1. Each tile contains an in-order, dual-issue UltraSPARC-III Cu processor with 2 GHz clock frequency. The first level cache is a split-cache with 4-way set-associative 64 KB instruction

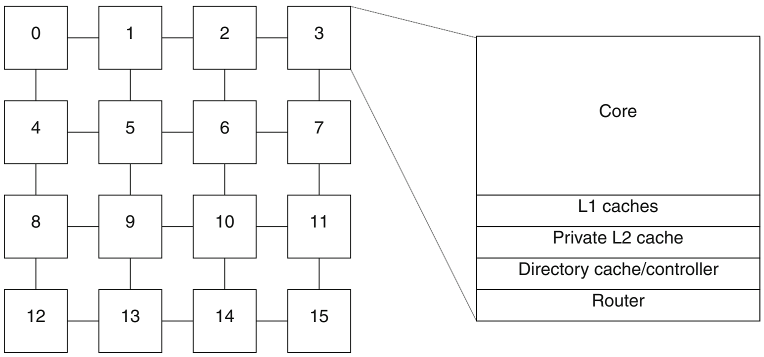

Fig. 1 Target architecture; 16-core on a 4 × 4 MESH topology

and data caches. Second level caches are private, unified 4-way set-associative 2 MB caches with 10-cycle hit latency. A directory is distributed among the tiles where each tile contains the blocks that are homed by that processor by utilizing a directory cache. The on-chip directory lookup latency is 20 cycles. Cores are connected with an unordered 2D-MESH with 1-cycle hop latency. If not otherwise specified, link bandwidth between tiles is assumed to be 8 bytes per cycle. Also, the network is assumed to operate at a frequency of one fourth of the CPU due to power and energy constraints.

3.2 Baseline Cache Coherence Protocol

In this work, we present two adaptive cache coherence protocols: PCA and PCBA. Both of these adaptive, hybrid protocols are based on a write-invalidate directory protocol with MOESI states. The Exclusive-clean (E) state obviates upgrade misses to non-shared data. Race conditions are resolved at the home node by using a busy/active state for each request. Order of arrivals at the home node establishes the order in which the racing requests are serviced [34]. Directory induced latency is a bottleneck in such a protocol. In addition, write-invalidate protocols tend to generate high cache-to-cache misses which can also further become a performance issue as the number of cores increases on a single-chip. Also, the same problem applies to multi-CMP systems. As such, the goal of the PCA protocol is to mitigate the latency burden driven by the cache-to-cache misses. The following section describes the implementation details of PCA and PCBA protocols.

4 Implementation Details

This section describes the details about the proposed cache coherence protocols (PCA and PCBA). To do so, producer-consumer sharing pattern detection and bandwidth-adaptive mechanisms are illustrated. In addition, implementation details are given to integrate such mechanisms into the proposed cache coherence protocols.

4.1 Producer-Consumer Sharing Pattern Detection

Producer-consumer sharing pattern can be defined via the following regular expression:

$$(W_a)(R_{\forall b:b\neq a})^m : a, b \in S,\ m \geq 1 \tag{1}$$

W_i and R_i refer to write and read operations by processor i. S refers to the complete list of processors in the system.

Algorithm 1 Algorithm to detect Producer-consumer Sharing

```
1: $pred\_cnt \leftarrow 0$
2: for all shared access to cache line A do
3:     if $request$ is exclusive then
4:         $sharers \leftarrow requester$                                ▷ Sharing Vector is reset
5:     end if
6:     if $request$ is shared then
7:         $sharers \leftarrow sharers \cup requester$
8:     end if
9:     if $request$ is shared or exclusive then
10:        if $requester \in recent\_sharers$ then
11:            $pred\_cnt \leftarrow min(pred\_cnt + 1, 7)$
12:        else
13:            $recent\_sharers \leftarrow recent\_sharers \cup requester$
14:        end if
15:    end if
16:    if $pred\_cnt = 7$ then
17:        Mark cache line A as a producer/consumer
18:    end if
19: end for
20:     ▷$pred\_cnt$ is referring to the Prediction Counter field in the directory cache
21:     ▷$pred\_cnt$ and $recent\_sharers$ are reset when the line is evicted from the directory cache
```

As illustrated in Expression 1, producer-consumer sharing pattern is defined in which one processor modifies a data block (producer) and a set of processors read the block (consumers). To detect cache lines that exhibit a producer-consumer sharing pattern, we extended each directory cache entry with a *recent_sharers* field to track the potential consumers as also suggested in [6]. In addition, a 3-bit saturating counter *prediction_counter* is used to track stable producer-consumer sharing pattern for a cache line. Figure 2 shows the directory cache entries extended with the *prediction_counter* and the *recent_sharers* fields. Since we experimented a directory-cache with 8K entries, this results in just 19 KB extra hardware cost per core to implement PCA. The *recent_sharers* field is analogous to the sharing vector (the *sharers*

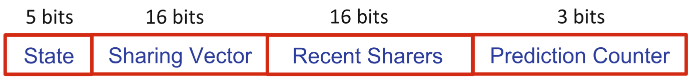

Fig. 2 Extended directory-cache fields

field) used by the directory and it includes a superset of the sharing processors with the addition of some other processors that recently accessed the same cache line. For each activity generated from the *recent_sharers* list, a 3-bit saturating *prediction_counter* is incremented. Also during each cache line replacement, the prediction history will be reset for that specific cache block. Tracking is only conducted for those lines cached by that specific core to minimize hardware requirements. Algorithm 1 demonstrates how we keep track of the recent history of sharers to predict potential consumers in the future. To simplify the pseudo-code for clarity, operations on *sharers* and *recent_sharers* fields are separated. As such, lines 2–8 show the operations on the *sharers* field and lines 9–15 show the operations on the *recent_sharers* field. Accordingly, for an exclusive access to a cache line, the *sharers* is reset as in a typical directory (as shown in line 4). However, an exclusive access does not result in a reset for the "*Recent Sharers* field (as shown in lines 9–15) in order to keep a history of sharers for this cache line. When the *prediction_counter* is saturated, the corresponding cache line is marked as a producer/consumer line. Thereby, future modifications to this line generate updates to the potential consumers identified by the *Recent Sharers* field.

4.2 Bandwidth Adaptive Mechanism

Write-update based protocols tend to generate heavy network traffic as each modification to a shared cache line will cause data updates to all sharers. Having this is mind, our proposed protocols trigger updates only for those lines that are exhibiting a producer-consumer sharing pattern. Thereby, in our proposed protocols, not only corresponding coherence misses are reduced but also the network traffic is kept low. However, there might be situations where either the network bandwidth is limited or a congestion is caused by a hot-spot in the network. Therefore, we propose a bandwidth-adaptive mechanism to alleviate the burden of potential updates in such scenarios. This bandwidth-adaptive mechanism is implemented on top of the PCA protocol resulting in the PCBA protocol.

A flowchart of how PCA and PCBA protocols are interrelated to each other and an overview of the underlying adaptive decision mechanisms can be seen in Fig. 3. To summarize, two important adaptive decision mechanisms are required: a producer-consumer (PC) detection mechanism and the bandwidth adaptive mechanism which is how PCBA extends PCA. PCBA is designed to control network traffic generated by the write-updates in PCA when the network has a limited bandwidth. PCBA utilizes a bandwidth adaptive mechanism that is inspired from an earlier study by Martin et al. [28], *BASH*, which dynamically adapts to the available bandwidth to switch between a broadcast-based protocol to a directory like protocol. However, we utilize bandwidth-adaptive framework as augmented to the PCA protocol to intelligently trigger write-update optimizations based on the available bandwidth. BASH uses a utilization counter in each core to calculate the link utilization and a policy counter to probabilistically decide whether to unicast or broadcast. The policy counter was mainly used to avoid oscillations in the system as BASH continuously needs to make decisions. However, our bandwidth adaptive policy comes only active for those cache blocks exhibiting producer-consumer sharing. Hence, we only implemented a signed

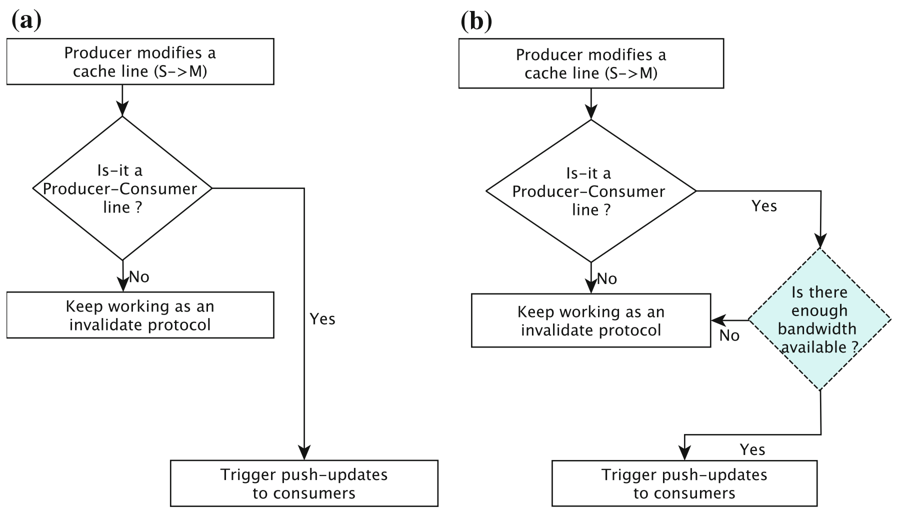
(a)
Producer modifies a cache line (S->M)
Is-it a Producer-Consumer line ?
No
Keep working as an invalidate protocol
Yes
Trigger push-updates to consumers
(b)
Producer modifies a cache line (S->M)
Is-it a Producer-Consumer line ?
Yes
No
Keep working as an invalidate protocol
No
Is there enough bandwidth available ?
Yes
Trigger push-updates to consumers

Fig. 3 Flowcharts comparing the PCA and PCBA protocols. **a** PCA ptotocol flowchart. **b** PCBA protocol flowchart

utilization counter over a sampling interval of 256 cycles to identify if the link utilization is above or below a *cut-off threshold* (75 %) as suggested in [28]. We also conducted a sensitivity analysis with varying static values between 25 and 95 %, and it was concluded that 75 % performs well in most cases as a threshold value. Experimental analysis which shows the importance of such a mechanism under high bandwidth utilization periods can be seen in Fig. 10.

4.3 Protocol Implementation

In order to sustain correctness and differentiate different scenarios cache coherence protocols utilize transition states. Each proposed protocol also inherits such transition states from the baseline protocol and used them to maintain correctness. In essence, for any cache line that does not exhibit a producer-consumer sharing pattern, our proposed protocols work like the baseline protocol which is basically enforcing a write-invalidate policy for all cache lines.

All of the proposed protocols can be implemented on top of any write-invalidate directory protocol, but for clarity we gave an overview of our baseline protocol in Sect. 3.2. In addition, Fig. 4 shows how a typical write-invalidate based protocol works for a sample producer-consumer data sharing example. Each arrow presents a hypothetical time line and designates a specific location in the system. From left to right, these locations are the producer's cache, the home tile which also includes the directory cache, the consumer's cache, and the main memory. The example assumes a single producer and a single consumer for clarity. Also, the cache line is assumed to be loaded and modified earlier so that the cache line actually exists in the producer's cache. Similarly, the cache line also exists in the directory cache of the home node/tile. Initial state of the cache line is OWNED in the producer's cache and SHARED in

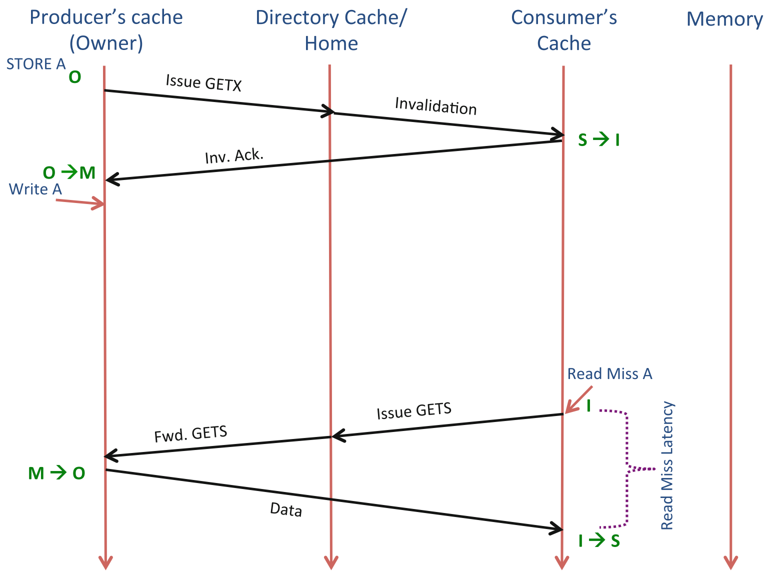

Fig. 4 Write-invalidate directory protocol

the consumer's cache. The directory cache is inside the home tile and this can be the same tile as the producer. This does not change the order of transactions but the latency of each message, and again for clarity it can be assumed that they represent different cores. Bold lines refer to coherence traffic and dashed gray lines correspond to avoided traffic. Figure 4 presents how a write-invalidate protocol would suffer a remote cache miss for a producer-consumer sharing cache line. These remote cache misses can become costly as the number of cores increases. Figures 5 and 6 present how prediction can help to avoid these coherence misses on the consumers. Figure 5 represents an earlier study by Cheng et al. [6] and our CMP specific optimizations can be seen by examining Fig. 6. Our proposed protocols use the same fundamentals to implement a data forwarding mechanism to trigger updates to consumers. However, Fig. 6 only mentions PCA for simplicity. In fact, differences between our proposed protocols come from interconnection resource aware mechanisms as explained earlier.

The initial transactions in all the examined protocols are same, which presents how the producer gets the exclusive rights on the cache line in order to modify it. With that modification, the cache line becomes MODIFIED in the producer's cache, and invalidated in the consumer's cache. After this, assuming the prediction mechanism detects that the cache line is actually a producer-consumer sharing line, consumers will be updated in Figs. 5 and 6. However, there is a major difference on how our proposed scheme and previous study handles the write-update process. Previous work [6] requires a WRITEBACK transaction after the producer modifies the cache line. As such, the cache line is written back to memory and in the meantime the consumer is updated. Even this scheme requires the producer to downgrade its rights on the cache line to be able to do the WRITEBACK, and then the home node updates the producer

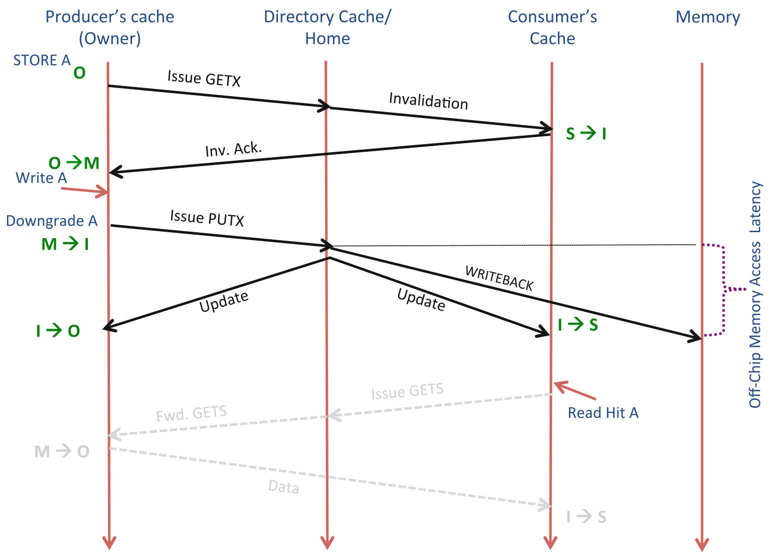

Fig. 5 Related work [6]

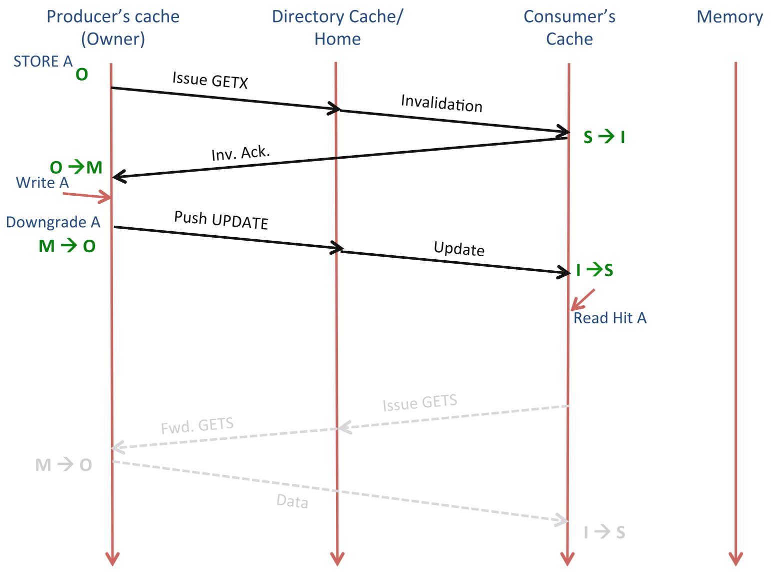

Fig. 6 PCA

along with the consumer. As we discussed earlier, CMPs provide unique optimization opportunities and avoiding off-chip memory accesses is definitely crucial to harness them. In Fig. 6, it is shown how PCA tries to avoid off-chip memory accesses wherever possible to exploit the on-chip resources. In addition, the way the update mechanism is triggered differs between the previous work and PCA.

More details about our trigger update mechanism can be found in the next section.

4.4 Trigger Update Mechanism

The *Trigger Update* event is integrated into the baseline cache coherence protocol to be able to intelligently update the consumer caches. This event becomes active after a certain delay when the producer modifies the producer-consumer cache line. This mechanism is analogous to the delayed intervention mechanism discussed in [6]. However, our trigger update mechanism improves the previous study by adding a dynamic mechanism to trigger updates. In this process, it is important to choose a long enough delay so that producers would finish their write-bursts. On the other hand, by selecting a long delay, it might be too late to update the consumers and we would not be able to prevent a cache miss in one of the consumer caches. Thereby, we did a sensitivity analysis which can be seen in Fig. 11 in order to select the best value. Accordingly, a static 1,000 cycle latency is selected to trigger the update mechanism after the producer modifies a cache line. However, this value should be configured based on the specific hardware configuration and system settings during the design stage as the optimal value may differ from the one chosen for our specific system.

As an optimization to this static trigger update mechanism, we devised a novel dynamic scheme that can adapt to the consumer's behavior in such update scenarios. This dynamic trigger mechanism still uses the static trigger delay, however, if there is a GETS (Get Shared) request from one of the consumers before the delay is ended, not only the GETS request is answered with the corresponding data block but also a *Trigger Update* event is called to push updates to the other consumers. In the meantime, this cancels the trigger delay to prevent unnecessary future updates. If the request comes from a core that is at the point not a member of the consumer's list, the requesting core is added to the consumer list and updates are still pushed to the consumers. As such, this dynamic scheme can improve the performance of the update mechanism by preventing late triggers that fail to prevent cache misses for the consumers. There is no difference between the static and dynamic schemes if the GETS request comes after the trigger delay period. In fact, that scenario is our goal which results in a cache hit in the consumer caches.

4.5 Verification of the Protocol

PCA and PCBA rely on the baseline coherence protocol correctness. However, extensions made to the base protocol can easily create race conditions which need to be verified and tested. For verification, we utilized the stress tests provided by GEMS. This synthetic testing mechanism generates excessive race conditions on the cache coherence protocol to identify potential coherence bugs. Furthermore, we managed to run all benchmark suites to completion with successful data verification at the end.

5 Experimental Analysis

This section presents our experimental study; simulation environment, workloads and performance evaluation of our proposed protocols.

Table 1 Architectural details

Parameter	Value
Processors	16 × UltraSPARC-III Cu processor cores, in-order, 2-way, 2 GHz
L1 Caches	Split I&D, 64 KB 4-way set associative with 64-byte blocks per core, 1-cycle hit latency
L2 Caches	4-way set associative with 64-byte blocks per core
Directory cache (DC)	8,192 entries, 20-cycle lookup latency
Cache replacement policy	Pseudo-LRU
Coherence mechanism	Directory-based protocol based on MOESI states
Memory	512 MB per core, shared memory, 160-cycle response latency
Interconnect	2D MESH (4 × 4), 1-cycle latency per hop

Table 2 Simulation workloads

Benchmark	Input data set
Barnes	16 K nodes, 123 seed
Cholesky	Input tk29.O
FFT	64 K complex data points
FMM	16 K particles
NPB BT	12*12*12 nodes, # of iterations = 60
NPB CG	Size = 1, 400, # of iterations = 15
NPB LU	16*16*16 nodes, # of iterations = 50
Ocean	258*258 grid, 10^{-7} error tolerance
Ocean_noncont	258*258 grid, 10^{-7} error tolerance

5.1 Simulation Environment

This research is conducted with full-system simulation utilizing Virtutech Simics [25] extended with a modified version of the GEMS toolset [26]. Simics provides the functional full-system simulation and GEMS provides a detailed memory system timing infrastructure which observes all cache coherence protocol messages and corresponding state transitions. The simulated machine is a 16-processor SPARC system running unmodified Solaris 10. Table 1 provides the details of the simulation environment along with the target architecture details. Results are reported only from the parallel computational phase of each workload.

5.2 Simulation Workloads

The benchmarks for the full-system simulation study were selected from the SPLASH-2 [37] benchmark suite and the NAS Parallel benchmark (NPB) suite [1]. The input data sets for the simulation workloads are provided in Table 2.

Barnes is an implementation of the Barnes-Hut method to simulate the interaction of a system of bodies over time in a gravitational system (N-body problem). BT solves a partial differential equation using a block tridiagonal algorithm. CG implements unstructured matrix vector multiplication via conjugate gradient method to approximate the smallest eigenvalue of a large, sparse, symmetric positive definite matrix. Cholesky kernel performs a blocked cholesky factorization on a sparse matrix. FFT benchmark solves a complex, one-dimensional version of the $radix-\sqrt{n}$ six-step FFT. FMM application implements a parallel adaptive Fast Multi-pole method to simulate the N-body problem like Barnes. LU benchmark refers to the LU decomposition. Ocean application is a simulation of large-scale ocean movements based on eddy and boundary currents. We used both of the contiguous and non-contiguous versions of the Ocean code. Non-contiguous version implements the grids in two-dimensional arrays which results in non-contiguous memory allocation. On the other hand, contiguous version implements the grids with three-dimensional arrays. First dimension is used to specify the local processor for a given partition and accordingly allows contiguous allocation of particles inside the local memory of processors that own them.

5.3 Results

In this section, we present performance analysis results from the workloads presented in Table 2. For all the figures; base refers to the baseline write-invalidate directory-based cache coherence protocol as described in Sect. 3.2. Figures 7 and 8 show the performance comparison between our proposed adaptive, hybrid protocol and the base. However, PCA uses two variants to differentiate the static trigger mechanism (PCA–ST) and the dynamic trigger mechanism (PCA–DT). PCBA is experimented with a static 1,000-cycle Trigger Update latency. Results show improvements up to 48 % with an average of 31 % for coherence misses as illustrated in Fig. 7. Coherence miss reduction is directly proportional to the available producer-consumer sharing pattern inside the given workload.

In general, our proposed optimizations provide performance gains proportional to the available producer-consumer sharing pattern inside the given workload. As such, Fig. 9 presents the percentage of LOADs that are part of a producer-consumer sharing pattern to better interpret the performance results. This profiling analysis is

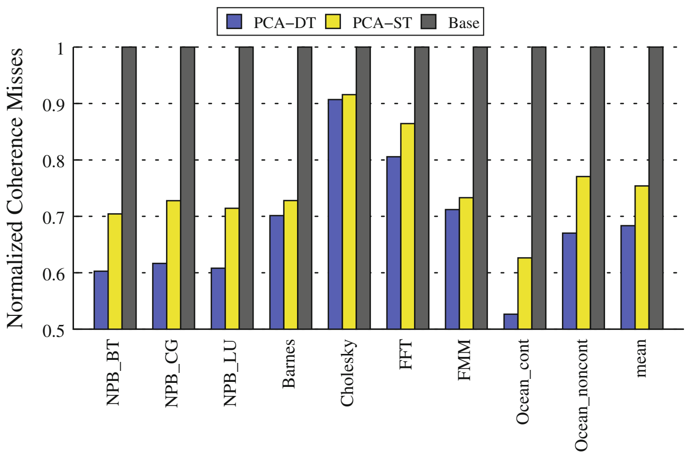

Fig. 7 Improvement in coherence misses

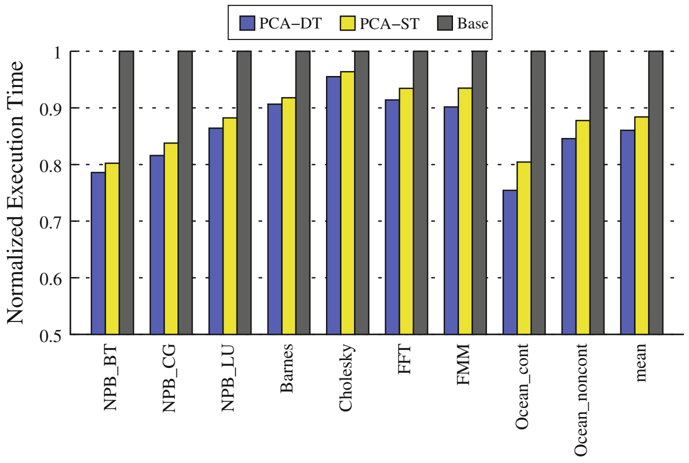

Fig. 8 Improvement in total execution time

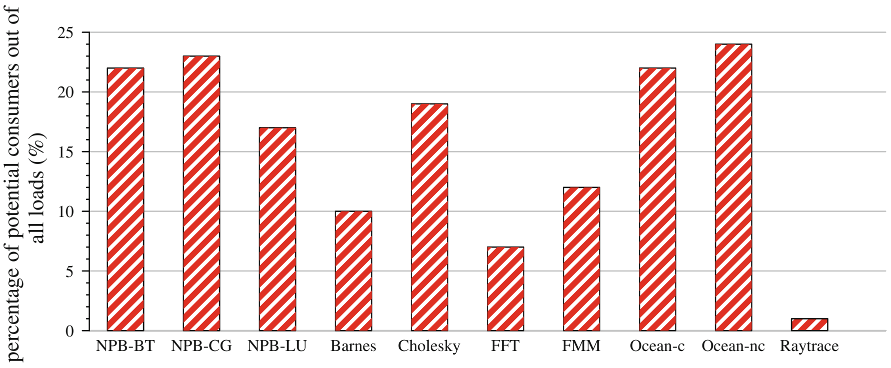

Fig. 9 Consumer pattern profiling

done by post-processing the memory traces of the applications obtained from our simulation framework. Total execution time improvements may vary for different architectural settings, but results demonstrate performance gains in all the applications experimented. Since we manage to reduce the coherence misses, these performance gains should even be higher for larger number of cores inside the chip where remote miss latencies will have more severe affects.

5.3.1 Bandwidth Adaptivity Analysis

Figure 10 shows results from our performance evaluation of the bandwidth adaptive mechanism. Results from four different workloads; NPB CG, NPB LU, Ocean, and Cholesky are presented with varying link bandwidth characteristics. Performance results for each link bandwidth value is normalized to the worst result for that specific link bandwidth. Experiments vary the static link bandwidth between 400 and 4,000 MB/s (The chip bi-directional cross-section bandwidth varying from 3.2 to

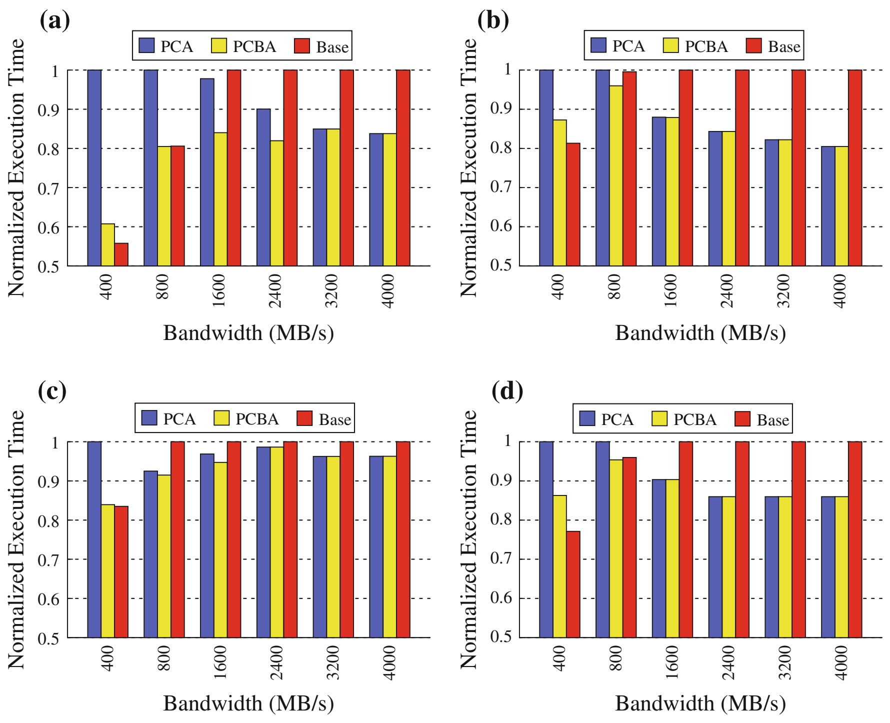

Fig. 10 Bandwith adaptivity results a NPB CG. b. ocean cont. c cholesky. d NPB LU

32 GB/s). One thing to note here is the fact that our simulator is based on an in-order SPARC core. Accordingly, bandwidth utilization in such a system is lower than most of the contemporary systems. To compensate this, we vary our tests in a way to throttle the links in terms of bandwidth to create a scenario that can happen with a more aggressive out-of-order processor under larger link bandwidths. The goal is to check whether PCBA can help when there is a network congestion in the system by dynamically adapting and cutting off any update optimizations which tend to generate more traffic than simple invalidates. Comparison of PCA, PCBA and base protocols clearly indicate how PCBA can adapt to low bandwidth situations and mitigate any potential performance loss. Results show that a hybrid/adaptive protocol such as PCA without a bandwidth-aware mechanism can suffer a performance degradation with high interconnect utilization. PCBA performed up to 40 % better in low-bandwidth scenarios as seen for the NBP CG case. Furthermore, when bandwidth is plentiful PCBA performs similarly to PCA. The bandwidth-adaptive mechanism mostly lets update optimizations to be triggered in such cases.

5.3.2 Sensitivity Analysis

In order to select the Trigger Update delay latency, we conducted a sensitivity analysis for a wide range of delay cycles while running our workloads. Corresponding results can be seen in Fig. 11. The *y* axes in the figures refer to the total execution time which

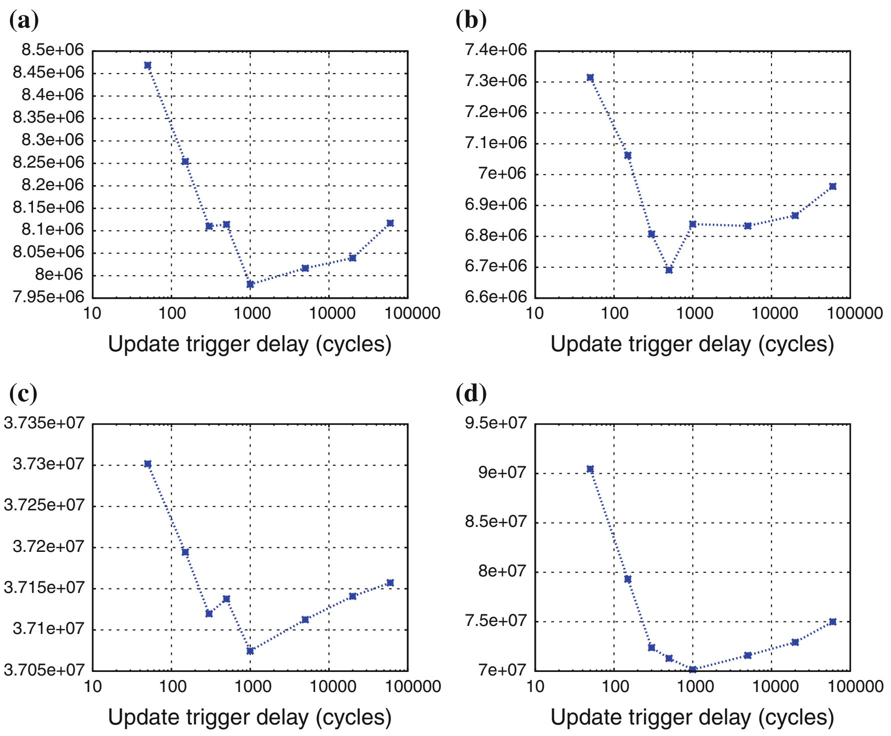

Fig. 11 Update trigger delay sensitivity analysis. **a** Barnes. **b** FMM. **c** LU. **d** Ocean cont

is presented as Ruby cycles. Ruby is the detailed memory timing simulator inside GEMS. On the *x* axes, different values, in cycles, for the update trigger delay are shown. Although there are slight variations, 1,000 cycles is proven to be an optimal value for most of the cases. However, some workloads tend to be insensitive for a wide range of delay cycles. Hence, they are not included in the results. In addition, our *dynamic trigger update* mechanism further optimizes this process, and makes the choice of 1,000 cycles much safer.

5.3.3 Scalability Analysis

There are some limitations with the simulation infrastructure we utilized to scale to large number of cores. Not only simulation times become extremeley long but also there are architecutal constraints that prevent such runs. However, in order to show results from larger number of cores, we developed a synthetic benchmark that exhibits producer-consumer sharing. Then, we ran this benchmark without SIMICS by just utilizing the GEMS memory timing simulator to get better understanding of the scalability results. Figure 12 shows results from this experiments. It can be seen that PCA outperforms the base write-invalidate protocol by 34 % while reducing the coherence misses 46 %.

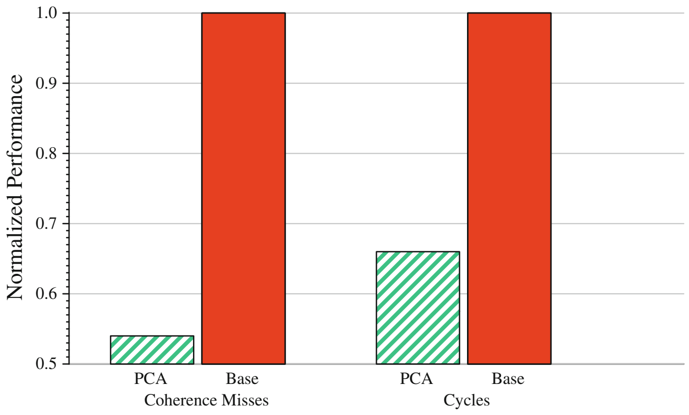

Fig. 12 Results from a 256 tiled-core CMP based on a producer/consumer synthetic benchmark

6 Conclusions and Future Work

Cache coherence has been investigated heavily in the research community, but most of the prior work targeted multiprocessor machines (MPs) comprised of multiple single-core processors. With the advent of CMPs, cache coherence becomes a first-order design issue at the chip level. Many constraints including interconnect, latency, power, and energy differ in MP and CMP designs. In addition, most contemporary cache coherence protocols use a write-invalidate policy which tends to generate a high number of coherence misses. This can become a major performance bottleneck as the number of cores inside a single chip increases. An adaptive cache coherence protocol enables a solution that can perform well in many system configurations and under different workloads. Thus, in this paper we propose a novel bandwidth adaptive, hybrid cache coherence protocol that is optimized in accordance with the CMP specific technological parameters to provide better performance. We presented adaptive mechanisms that can reduce coherence misses as well as can adapt to varying interconnect utilization to avoid network congestions. Furthermore, the extra hardware cost needed to implement such an adaptive mechanism is kept low at around 19 KB per core.

Experimental evaluation is conducted on a many-core tiled CMP via full-system simulation. Performance analysis is presented on a set of scientific applications from the SPLASH-2 and the NAS parallel benchmark suites. Results showed that the proposed mechanisms reduce cache-to-cache misses up to 48 % by an average of 25 % which in turn improves application performance up to 25 % by an average of 14 %. The performance benefits have more impact as the number of cores increases which shows 34 % performance improvement while running a producer-consumer synthetic benchmark under a 256-core tiled CMP. In addition, bandwidth adaptive hybrid protocol is proven to perform well under varying interconnect utilizations. For low-bandwidth scenarios, our proposed protocol prevents any performance loss that might happen with the write-update optimizations. In addition, when bandwidth is plentiful our proposed solution outperforms the base directory protocol. One area of future work is to extend this study on larger CMPs. Hierarchical policies can also be integrated to accommodate the emerging many core systems and multi-CMP platforms. Additionally, more aggressive prediction mechanisms can be evaluated.

Acknowledgments Authors would like to thank Dan Gibson from Google, formerly at University of Wisconsin Multifacet group, for his help and suggestions on our implementations in GEMS simulation infrastructure. Authors also would like to thank Arctic Region Supercomputing Center (ARSC) for their support in this research.

References

1. NAS Parallel Benchmarks: OpenMP version developed by the Omni group http://www.hpcs.cs.tsukuba.ac.jp/omni-openmp
2. Acacio, M., González, J., García, J., Duato, J.: Owner prediction for accelerating cache-to-cache transfer misses in a cc-NUMA architecture. In: Proceedings of the 2002 ACM/IEEE Conference on Supercomputing, pp. 1–12. IEEE Computer Society Press Los Alamitos, CA, USA (2002)
3. Acacio, M.E., González, J., García, J.M., Duato, J.: The use of prediction for accelerating upgrade misses in cc-NUMA multiprocessors. In: IEEE PACT, pp. 155–164. IEEE Computer Society (2002)
4. Alam, S.R., Barrett, R.F., Kuehn, J.A., Roth, P.C., Vetter, J.S.: Characterization of scientific workloads on systems with multi-core processors. In: IISWC, pp. 225–236. IEEE (2006)
5. Anderson, C., Karlin, A.R.: Two adaptive hybrid cache coherency protocols. In: International Symposium on High-Performance Computer Architecture (HPCA), pp. 303–313 (1996)
6. Cheng, L., Carter, J.B.: Extending cc-numa systems to support write update optimizations. In: SC '08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, p. 30. IEEE/ACM (2008)
7. Chu, M., Ravindran, R., Mahlke, S.: Data access partitioning for fine-grain parallelism on multicore architectures. In: MICRO '07: Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 369–380. IEEE Computer Society, Washington, DC, USA (2007). doi:10.1109/MICRO.2007.11
8. Cox, A.L., Fowler, R.J.: Adaptive cache coherency for detecting migratory shared data. In: International Symposium on Computer Architecture (ISCA), pp. 98–108 (1993)
9. Dahlgren, F.: Boosting the performance of hybrid snooping cache protocols. In: ISCA '95: Proceedings of the 22nd Annual International Symposium on Computer Architecture, pp. 60–69. ACM, New York, NY, USA (1995). doi:10.1145/223982.223998
10. Dahlgren, F., Stenström, P.: Reducing the write traffic for a hybrid cache protocol. In: International Conference on Parallel Processing (ICPP), pp. 166–173 (1994)
11. Eggers, S.J., Katz, R.H.: Evaluating the performance of four snooping cache coherency protocols. SIGARCH Comput. Archit. News **17**(3), 2–15 (1989). doi:10.1145/74926.74927
12. Eisley, N., Peh, L.S., Shang, L.: In-network cache coherence. In: MICRO 39: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 321–332. IEEE Computer Society, Washington, DC, USA (2006). doi:10.1109/MICRO.2006.27
13. Eisley, N., Peh, L.S., Shang, L.: Leveraging on-chip networks for data cache migration in chip multiprocessors. In: PACT '08, pp. 197–207. ACM, New York, NY, USA (2008). doi:10.1145/1454115.1454144
14. Fensch, C., Cintra, M.: An OS-based alternative to full hardware coherence on tiled CMPs. In: 14th International Symposium on High Performance Computer Architecture (HPCA), pp. 355–366. IEEE (2008). doi:10.1109/HPCA.2008.4658652
15. Geer, D.: Industry trends: chip makers turn to multicore processors. IEEE Comput. **38**(5), 11–13 (2005)
16. Ghosh, D., Carter, J.B., III, H.D.: Perceptron-based coherence predictors. In: Proceedings of 2nd Workshop on Chip Multiprocessor Memory Systems and Interconnects (CMP-MSI), in Conjunction with ISCA 2008 (2008)
17. Gorder, P.F.: Multicore processors for science and engineering. Comput. Sci. Eng. **9**(2), 3–7 (2007). doi:10.1109/MCSE.2007.35
18. Grahn, H.K., Stenström, P.: Evaluation of a competitive-update cache coherence protocol with migratory data detection. J. Parallel Distrib Comput **39**, 39–42 (1996)
19. Karlin, A.R., Manasse, M.S., Rudolph, L., Sleator, D.D.: Competitive snoopy caching. Algorithmica **3**, 77–119 (1988)

20. Kaxiras, S., Goodman, J.R.: Improving cc-NUMA performance using instruction-based prediction. In: International Symposium on High-Performance Computer Architecture (HPCA), pp. 161 (1999)
21. Kaxiras, S., Young, C.: Coherence communication prediction in shared-memory multiprocessors. In: International Symposium on High-Performance Computer Architecture (HPCA), pp. 156–167 (2000)
22. Kayi, A., Kornkven, E., El-Ghazawi, T., Al-Bahra, S., Newby, G.: Performance evaluation of clusters with ccNUMA nodes: a case study. In: HPCC '08, pp. 320–327 (2008)
23. Lai, A.C., Falsafi, B.: Memory sharing predictor: The key to a speculative coherent dsm. In: ISCA '99: Proceedings of the 26th, Annual International Symposium on Computer Architecture, pp. 172–183 (1999)
24. Leventhal, S., Franklin, M.: Perceptron based consumer prediction in shared-memory multiprocessors. In: ICCD 2006: International Conference on, Computer Design, pp. 148–154 (2006). doi:10.1109/ICCD.2006.4380808
25. Magnusson, P.S., Christensson, M., Eskilson, J., Forsgren, D., Hållberg, G., Högberg, J., Larsson, F., Moestedt, A., Werner, B.: Simics: a full system simulation platform. IEEE Comput. **35**(2), 50–58 (2002)
26. Martin, M.M.K.: Formal verification and its impact on the snooping versus directory protocol debate. In: ICCD 2005: International Conference on Computer Design, pp. 543–449. IEEE Computer Society (2005)
27. Martin, M.M.K., Harper, P.J., Sorin, D.J., Hill, M.D., Wood, D.A.: Using destination-set prediction to improve the latency/bandwidth tradeoff in shared-memory multiprocessors. In: International Symposium on Computer Architecture (ISCA), pp. 206–217. IEEE Computer Society (2003)
28. Martin, M.M.K., Sorin, D.J., Hill, M.D., Wood, D.A.: Bandwidth adaptive snooping. In: International Symposium on High-Performance Computer Architecture (HPCA), pp. 251–262 (2002)
29. Marty, M.R., Bingham, J.D., Hill, M.D., Hu, A.J., Martin, M.M.K., Wood, D.A.: Improving multiple-cmp systems using token coherence. In: ISCA '05: Proceedings of the 32nd Annual International Symposium on Computer Architecture, pp. 328–339. IEEE Computer Society (2005)
30. Moore, G.E.: Cramming more components onto integrated circuits. Electronics **38**(8), 114–117 (1965)
31. Mukherjee, S.S., Hill, M.D.: Using prediction to accelerate coherence protocols. In: International Symposium on Computer Architecture (ISCA), pp. 179–190 (1998)
32. Nilsson, H., Stenström, P.: An adaptive update-based cache coherence protocol for reduction of miss rate and traffic. In: Proceedings of Parallel Architectures and Languages Europe (PARLE), pp. 363–374. Springer (1994)
33. Nilsson, J., Landin, A., Stenström, P.: The coherence predictor cache: a resource-efficient and accurate coherence prediction infrastructure. In: IPDPS '03: Proceedings of the International Parallel and Distributed Processing Symposium, p. 10. IEEE Computer Society (2003)
34. Raghavan, A., Blundell, C., Martin, M.M.K.: Token tenure: patching token counting using directory-based cache coherence. In: MICRO, pp. 47–58. IEEE Computer Society (2008)
35. Raynaud, A., Zhang, Z., Torrellas, J.: Distance-adaptive update protocols for scalable shared-memory multiprocessors. In: HPCA '96: Proceedings of the Second International Symposium on High-Performance Computer, Architecture, pp. 323–334 (1996). doi:10.1109/HPCA.1996.501197
36. Stenström, P., Brorsson, M., Sandberg, L.: An adaptive cache coherence protocol optimized for migratory sharing. In: International Symposium on Computer Architecture (ISCA), pp. 109–118 (1993)
37. Woo, S.C., Ohara, M., Torrie, E., Singh, J.P., Gupta, A.: The SPLASH-2 Programs: characterization and methodological considerations. In: ISCA '95, pp. 24–36 (1995)
38. Yeh, T.Y., Patt, Y.N.: Alternative implementations of two-level adaptive branch prediction. In: International Symposium on Computer Architecture (ISCA), pp. 124–134 (1992)