# An Adaptive Cache Coherence Protocol for Chip Multiprocessors

Abdullah Kayi
Department of Electrical and Computer Engineering
The George Washington University
Washington, DC 20052, USA
apokayi@gwmail.gwu.edu

Tarek El-Ghazawi
Department of Electrical and Computer Engineering
The George Washington University
Washington, DC 20052, USA
tarek@gwu.edu

## ABSTRACT

Multi-core architectures also referred to as Chip Multiprocessors (CMPs) have emerged as the dominant architecture for both desktop and high-performance systems. CMPs introduce many challenges that need to be addressed to achieve the best performance. One of the big challenges comes with the shared-memory model observed in such architectures which is the cache coherence overhead problem. Contemporary architectures employ write-invalidate based protocols which are known to generate coherence misses that yield to latency issues. On the other hand, write-update based protocols can solve the coherence misses problem but they tend to generate excessive network traffic which is especially not desirable for CMPs. Previous studies have shown that a single protocol approach is not sufficient for many sharing patterns. As a solution, this paper evaluates an adaptive protocol which targets write-update optimizations for producer-consumer sharing patterns. This work targets a minimalistic hardware extension approach to test the benefits of such adaptive protocols in a practical environment. Experimental study is conducted on a 16-core CMP by using a full-system simulator with selected scientific applications from SPLASH-2 and NAS parallel benchmark suites. Results show up to 40% improvement for coherence misses which corresponds to 15% application speedup.

## Keywords

chip-multiprocessors, cache coherence protocols, multi-core architectures, directory-based cache coherence

## 1. INTRODUCTION

The demand for computing power continues to increase in virtually every domain, from the basic desktop systems to the high-end computing platforms. In the past, performance increase in processors was mainly reached by increasing clock frequency and designing more complex systems [9, 49]. As stated in Moore's law, the number of transistors inside a single chip has continued to increase exponentially. However, using all these transistors for a single processing core is facing practical challenges. These challenges include power dissipation, thermal constraints and limited instruction-level parallelism [10, 24]. Accordingly, 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). These architectures provide a solution to increase the performance capability on a single chip without requiring a complex system and increasing the power requirements [10, 13, 19, 26]. Therefore, CMPs have become the dominant architecture for both desktop and high-performance platforms. Several chip manufacturers including AMD, IBM, Intel, and Sun have released systems, which have multiple processing units on a single chip. This trend will likely to continue with more and more cores being put on a single chip [26, 3, 4]. Accordingly, not only the high-end computing systems but also the commodity computing platforms include a certain level of shared memory system with multiple processing cores.

Shared memory systems provide various advantages including a simple programming model but this comes with the cost of memory consistency and coherency problems. As we see more and more cores are put on a single-chip, cache coherency strategy will become a key performance bottleneck in CMPs as it was experienced in earlier SMP/ccNUMA systems. Recent research has shown the severe effects of cache coherence overhead on application performance including large high-performance clusters with multi-socket CMP based shared memory nodes [32].

Based on the policy during a write operation on shared data, cache coherence protocols fall into two main categories, *write-update* based protocols or *write-invalidate* based protocols. Write-update protocols entail interprocessor communication on every write operation to shared data. Write-invalidate protocols maintain cache coherence by invalidating copies of a memory block when the block is modified by a processor. The advantage of this is that at most the first write, in a sequence of writes to the same block with no intervening read operations from other processors, causes inter-processor communication. Hence, subsequent write operations can be completed locally until the same data block is requested by another processor. As such, write-invalidate protocols yield to less network traffic and accordingly pro-


*IFMT* '10, 19-JUN-2010, Saint-Malo, France

vide a better scalable solution to the cache coherence problem but they suffer from the high read latency induced by cache coherence misses. On the other hand, previous research has shown that coherence misses or cache-to-cache misses comprise a big portion of overall cache misses in shared memory systems [7, 6, 23, 18, 43] and write-update protocols can degrade these coherence misses [50, 27, 17]. However, write-update protocols tend to generate high traffic on the interconnect due to unnecessary updates especially for migratory objects. So, a single cache coherence policy solution to effectively provide performance for different type of applications is not feasible. Having said that, it is difficult to implement a write-update based protocol on unordered interconnects and in general performance improvements degrade by having a sequentially consistent memory system for these protocols. Therefore, most contemporary systems implement a write-invalidate based cache coherence protocol.

As a solution, we propose a data-forwarding protocol for producer-consumer sharing blocks on top of a write-invalidate protocol similar to the study conducted by Liqun et. al for ccNUMA systems in [17, 16]. The goal is to have a minimalistic approach for hardware requirements and not entail large design space modifications. As such, the proposed mechanisms do not change the memory model seen by the programmer and compiler. Proposed adaptive protocol can be implemented on top of any write-invalidate protocol and can work with existing language, compiler, and processor designs.

The rest of this paper is organized as follows: Section 2 gives an overview of the related work; Section 3 discusses some background information on the adapted base architecture details and the base directory cache coherence protocol. Section 4 describes the implementation details including the proposed Producer-Consumer Predictor Cache(PCPC), the consumer set prediction mechanism, and verification of the adaptive protocol. The simulator environment as well as the simulation workloads are explained in Section 5. Also, Section 5 presents and analyzes the experimental results; and Section 6 contains a summary and conclusions of the study with future work.

## 2. RELATED WORK

There have been numerous efforts to improve cache coherence protocols performance. The selection of the cache coherence write policy on shared data, write-invalidate versus write-update, has been the main focus of many earlier studies [29, 20, 51, 45, 22, 21, 33, 12, 50, 47]. Many papers have examined shared memory behavior and tried to optimize underlying cache coherence protocols for specific static data sharing patterns such as migratory sharing [20, 51], read-modify-write sequences [46, 47], pairwise-sharing [28], and producer-consumer sharing [17, 16]. This work targets the producer-consumer sharing as this sharing pattern was shown to be the weakness of write-invalidate protocols.

Broadcast-based snooping protocols versus directory-based protocols have also been examined to study the bandwidth and latency trade-off between these two choices [14, 41, 42]. Although snooping protocols provide lower latencies as targeted in this paper, they also generate heavy traffic and put too much burden on the interconnect. Furthermore, they usually require an ordered network such as bus interconnect which can be another performance bottleneck. As such, bus-based systems are not scalable. Hence, in this work we implemented our protocol utilizing a directory scheme.

Producer-initiated communication was suggested to alleviate long miss latencies. Koufaty *et al.* studied data forwarding mechanisms utilizing compilers [33], Abdel-Shafi *et al.* evaluated fine-grain producer-initiated communication in cache-coherent multiprocessors [5], Byrd *et al.* examined the performance gap between shared memory and message passing machines and analyzed producer-initiated communication as a solution to close this gap [15]. All these studies were at the software level, our study looks at the problem from hardware perspective and implements the proposed mechanisms as a hardware-based cache coherence protocol.

Dynamic self-invalidation was proposed by Lebeck and Wood [36] to eliminate the invalidation overhead. Later, Lai and Falsafi [35] improved this mechanism using last-touch prediction.

A different approach to deal with cache coherence overhead is based on coherence predictors. Mukherjee and Hill [44] proposed Cosmos coherence message predictor with an extended study on Yeh and Patt's two-level PAp branch predictor [53]. Kaxiras *et al.* [30, 31] 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.* [34] proposed pattern-based memory sharing predictors which reduces the memory requirements of the previous proposed predictors. Acacio *et al.* [7] 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 [8]. Coherence predictor cache was designed and evaluated in [48] as a resource-efficient coherence predictor. Nilsson *et al.* 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 which inspired us to use the same strategy for our Producer-Consumer Predictor Cache(PCPC). Martin *et al.* [40] 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 in [37, 25]. Most of these extensions require major modifications and relatively costly for a CMP design.

## 3. BACKGROUND INFORMATION

This section gives some details about the experimented base CMP architecture. Also, the governing base cache coherence protocol is described in this section.

### 3.1 Base Architecture

Our target system architecture is a tiled-CMP with 16 cores designed via 4x4 2D-MESH interconnect topology as can be seen in Figure 1. Each tile contains an in-order, dual-issue UltraSPARC-III Cu processor with 2GHz clock frequency. The first level cache is a split-cache with 4-way

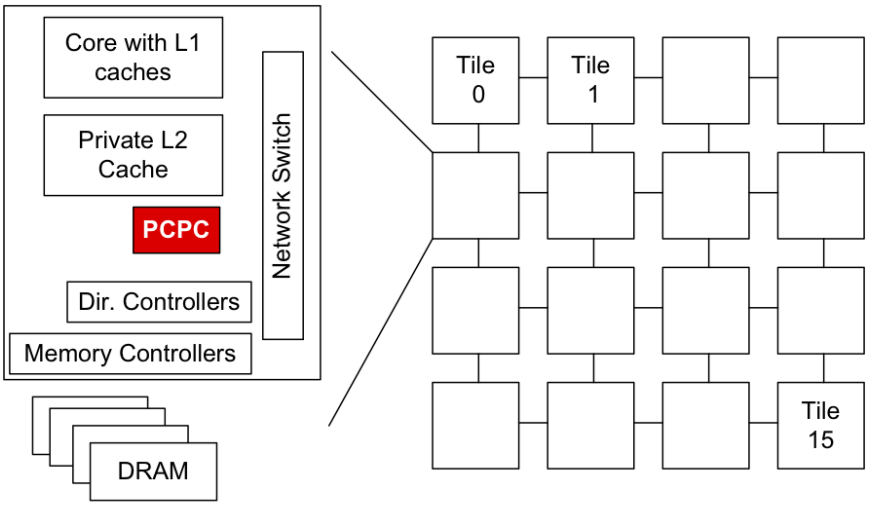


**Figure 1: Target Tiled-CMP architecture; 16 core-CMP connected via 4x4 2D-MESH topology**

set-associative 64KB instruction and data caches. Second levels caches are private, unified 4-way set-associative 2MB caches with 4-cycle hit latency. Directory is distributed among the tiles where each tile contains the blocks that are homed by that processor. Both the memory and directory is accessed via on-chip memory controllers. The corresponding tiled-CMP provides a globally addressable shared memory yielding to a cache-coherent Non-Uniform Memory Access(ccNUMA) architecture on a single chip. Cores are connected with an unordered 2D-MESH with an 8GB/s unidirectional link bandwidth. Table 1 presents the details of the target machine. Only addition to the base architecture to implement the proposed adaptive cache coherence protocol is the Producer-Consumer Predictor Cache(PCPC) which will be described in more details in the next section. Although, we adopted this tiled architecture as a base, our proposed work can work on any CMP with private caches at some level of the memory subsystem.

## 3.2 Base Coherence Protocol

Our adaptive protocol is based on a write-invalidate directory protocol with MOESI states. MOESI protocol is named after the five states possible for any line in a particular cache. That is, any given cache block can be at one of the following states:

1. Invalid(I): Data inside the corresponding cache line is stale, invalid.
2. Modified(M): Processor holds the most recent copy and no other processor has a copy.
3. Exclusive(E): Processor holds the most recent, correct copy of the data. Main memory also has a valid copy.
4. Shared(S): Similar to Exclusive state except multiple processors can have the recent copy. In order to have the recent copy also in the main memory, no other cache line may hold the data in the Owned state.
5. Owned(O): Only one cache line can be at this state, the other processors must be in the Shared state. This helps to identify the processor which is responsible for write-backs to main memory.

In a directory-based protocol, mainly following classification is utilized to differentiate processors [7]:

- The *home node* is the processor in whose main memory the requested block resides.
- The *exclusive node* is the processor that holds the most recent copy of the cache block.
- The *requesting node* is the processor that requests the cache line after a miss in its own cache.

The main disadvantage of a such a directory-based write-invalidate protocol is the 3-hop misses required to complete the events that are generated after L2 misses in requesting nodes in which home node doesn't have the latest copy. Our proposed protocol targets this bottleneck by alleviating the latency burden via data-forwarding protocol extensions to establish write-updates for lines that are deemed to be exhibiting producer-consumer sharing pattern.

# 4. IMPLEMENTATION DETAILS

In this section, we will give some more details about the proposed protocol and the hardware mechanisms to identify producer-consumer sharing pattern.

The ideal situation in the destination-set prediction is to speculate the most accurate set of receivers in a timely manner to alleviate overall latencies without causing extra traffic on the network. Unfortunately, accuracy comes with extra hardware cost. Our goal is a minimalistic approach to the problem to be able to provide reasonable extensions

**Table 1: Architectural Details**

| Parameter | Value |
|---|---|
| Processors | In-order 2-way UltraSPARC-III Cu processor, 2GHz |
| # of cores | 16 |
| L1 Caches | Split I&D, 64KB 4-way set associative with 64-byte blocks per core, 1-cycle hit latency |
| L1 Replacement Policy | Pseudo-LRU |
| L2 Caches | 4-way set associative with 64-byte blocks per core, 4-cycle hit latency |
| L2 Replacement Policy | Pseudo-LRU |
| PCPC Replacement Policy | Pseudo-LRU |
| Coherence Mechanism | directory-based protocol based on MOESI states |
| Memory | 512MB per core shared memory |
| Interconnect | 4x4 2D MESH, 4-cycle latency per hop, 8GB/s unidirectional link bandwidth |

to upcoming CMP systems. As we also mentioned before, large design space modifications is not desirable for our purposes. Thus, we implement the prediction mechanism on each core's directory controller.

## 4.1 Producer-Consumer Predictor Cache

Producer-Consumer Predictor Cache(PCPC) is the only hardware extension required to implement the proposed adaptive protocol. It uses the directory state information with some additional bits to track down the producer-consumer sharing pattern for a cache block. Each core has its own PCPC tracking only the blocks homed by that node. PCPC is tagged, 4-way set-associative and indexed by data block addresses with a size of 4K elements. PCPC utilizes address-filtering as suggested in [48] to eliminate the unnecessary caching of blocks to save design space. Only blocks that are detected as coherence blocks are cached in PCPC. To determine this, PCPC utilizes a *last_writer* bit to check during a cache miss whether the reader matches the last writer. Lines with different readers are selected as coherence blocks. Coherence prediction tracking is done parallel with the cache access. Each predictor entry also includes a 3-bit saturating consumer counter to track the consumers. Although there are lots of limitations with this approach, this study aims to analyze potential gains with such small hardware extensions. We are also exploring more aggressive approaches as an ongoing research.

The adaptive protocol is based on a write-invalidate protocol with MOESI states as described earlier in Section 3.2. Protocol is extended to support data forwarding as if in a write-update protocol to the predicted consumer set when prediction mechanism detects a producer-consumer sharing pattern. All memory system is based on a sequentially consistent system which makes it a more practical solution although relaxed-memory consistency models would have helped our system architecture better in terms of performance.

## 4.2 Producer-Consumer Prediction

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

$$... + (W_a)(R_{\forall b:b\neq a}) + (W_x)(R_{\forall y:y\neq x}) + ...a, b, x, y \in S \quad (1)$$

$W_x$ and $R_x$ refers to write and read operations by processor $x$. $S$ refers to the complete list of processors in the system.

In order to track-down the producer-consumer sharing pattern we follow a similar approach as in [16, 40]. PCPC is extended with *recent_sharers* field to designate the potential consumers. The *recent_sharers* field is similar to sharing vector used by the directory and it includes a superset of sharing processors with the addition of some other processors that accessed the same cache line. In essence, it represents the consumer set for that cache block. Here is how the algorithm works for producer-consumer sharing pattern detection.

For each shared or exclusive request generated from the *recent_sharers* list, consumer-counter is incremented. For requests from other processors consumer-counter is decremented. Also, with cache line replacement prediction history will be reset for that specific cache block.

## 4.3 Verification of the Protocol

Formal verification of large scale cache coherence protocols in a reasonable environment is a big research still being worked on due to the complex nature of cache coherence protocols. Although formal correctness verification of the proposed protocol is an ongoing work, we utilized the stress testing mechanism provided with the GEMS simulator[39]. Basically, this was designed to stress test the coherence protocols with excessive race conditions. For example, one of the tests that we utilized heavily generates race conditions by issuing multiple exclusive writes to same cache lines. This synthetic testing mechanism helped us to identify the 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 describes the experimental study; simulation environment, workloads and the corresponding observed results.

## 5.1 Simulator Environment

This research is conducted with full-system simulation utilizing Virtutech Simics [38] extended with the GEMS toolset [39]. A full-system simulator was chosen to include operating system effects to establish a more practical platform for our measurements. 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. In order to simulate only the computational part, breakpoints are inserted into the benchmarks and checkpointing is used. Furthermore, caches are warmed up to avoid cold misses as suggested in [11].

## 5.2 Simulation Workloads

The benchmarks for the full-system simulation study were selected from the SPLASH-2[52] benchmark suite and Omni's OpenMP version of the NAS Parallel benchmark(NPB) suite [1, 2]. The input data sets for the simulation workloads are provided in Table 2.

**Table 2: Benchmarks and input data sets**

| Benchmark | Input Data Set |
|---|---|
| *barnes* | 16384 nodes, 123 seed |
| *cg* | Size=1400, # of Iterations=15 |
| *cholesky* | input tk29.O |
| *fft* | 64K complex data points |
| *fmm* | 16K particles |
| *lu_cont* | 512x512 matrix |
| *lu_noncont* | 512x512 matrix |
| *mg* | Size=32x32x32, # of Iteraritons=4 |
| *ocean_cont* | 258*258 grid, 1e-7 error tolerance |
| *ocean_noncont* | 258*258 grid, 1e-7 error tolerance |
| *radiosity* | -batch -room |
| *raytrace* | car.env |

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). 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, both contiguous and non-contiguous block allocation versions were experimented. MG is a 3D multi-grid benchmark that calculates an approximate solution $u$ to the discrete Poisson problem $\nabla u^2 = v$ with periodic boundary conditions. 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. Radiosity application computes the equilibrium distribution of light in a scene using the hierarchical diffuse radiosity method. Raytrace application implements an algorithm to render a three-dimensional scene onto a two-dimensional image plane using optimized ray tracing.

## 5.3 Results

In this section, we provide results from the experiments we conducted on a 16-core simulated CMP as described in Section 3.1. The results are reported from the parallel sections of each workload by utilizing the checkpointing mechanism in [38]. We collected traces from each of the simulated workloads using GEMS and SIMICS to do further post-processing to get more profile information about the sharing patterns of each workload. The collected traces directly correspond to the timed part of our simulations which refer to the parallel sections of the benchmarks. However, for all the runs the caches are warmed-up with an earlier collected traces to prevent cold-misses.

Figure 2 represents the upper-bound for the consumer prediction mechanism. It shows the percentage of all potential consumers out of all the LOAD operations observed and consumers are detected based on the regular expression defined in Equation 1. Since our optimization depends on the availability of such producer-consumer sharing sets, this analysis can help us evaluating the performance of the proposed protocol better. We tried to include a variety of workloads with different sharing patterns not just the applications with major producer-consumer sharing patterns. Raytrace, Barnes, and non-contiguous LU decomposition results show that we have a limited opportunity for possible optimizations with the consumer prediction mechanism. CG, Cholesky, MG, Ocean, and Radiosity have more than 10% of their LOADs as consumers whereas FFT, FMM, and contiguous LU decomposition have around 5% consumers out of all the LOADs observed. The least amount of consumers ratio was observed in Barnes, non-contiguous LU with 2%, and in Raytrace with less 0.5%.

We also analyzed the traces to generate a histogram of the length of the consumer set for each producer. Figure 3 presents the corresponding results by illustrating the size of potential consumer sets. Most of the applications are dominated by producers with a single consumer except Cholesky and both LU versions. If we classify the results even further; Barnes, FMM, and Radiosity shows results in between these two cases by having a single consumer per producer 60% of the time. Although this profiling information is an important aspect to evaluate the performance of the optimizations included in our proposed protocol, it is crucial to note that these results reveal high level profiling analysis without considering the actual identity of the consumers. In accordance to the size of the consumer sets, number of producers per memory block is also an important factor which can affect the prediction accuracy of the prediction mechanism. In general, multiple producers scenario is harder to track compared to a single producer case. Figure 4 illustrates the histogram of the number of producers per memory block. Again, most of the workloads reveal a single producer case except MG and Radiosity. Barnes and FMM exhibit similar patterns for both the producer size and the consumer size histograms as they have similar computational characteristics. Non-contiguous Ocean application have either single producer or

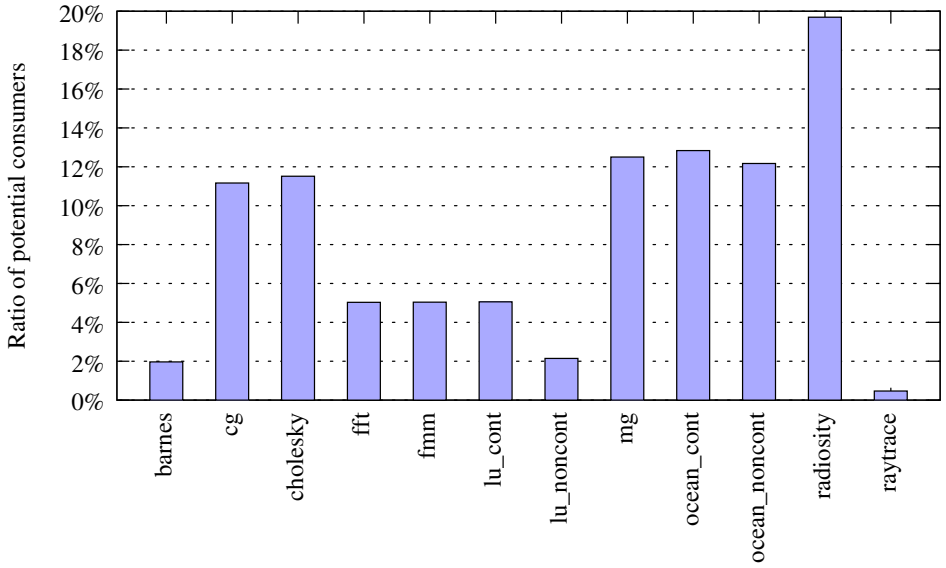


**Figure 2: Percentage of potential consumers out of all LOADs observed**

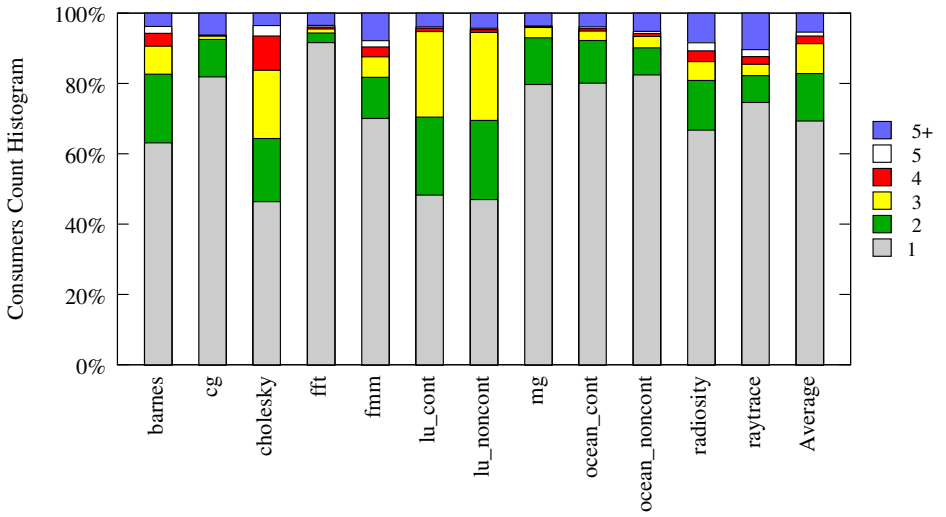


**Figure 3: Histogram of the number of consumers per producer**

two producers for 98% of all the producer-consumer sets.

Figures 5, 6 show results from the *adaptive* protocol compared with the *base* MOESI invalidate directory protocol. All results are normalized to *base* protocol values. Main goal of this work is to alleviate the *coherence misses* overhead in write-invalidate protocols. Corresponding results can be found in figure 5 which shows up to 40% reduction in coherence misses which was the case for the Ocean with contiguous allocation benchmark. Figure 6 illustrates the values for the total execution time measured for the parallel section of the workloads. Similarly contiguous Ocean performance is improved by 15% representing a case for the importance of coherence misses in overall application performance. Also, it proves the effectiveness of the proposed optimization via an adaptive cache coherence protocol.

Raytrace has the overall least performance improvement as it exhibits almost no producer-consumer sharing pattern. On the other hand, Radiosity was shown to have the largest ratio of potential consumers. However, it includes varying multiple producers per memory block and overall it was implemented with complex dynamic structures which limited the performance gains obtained from this application. Having said that, we still have 20% coherence misses reduction and almost 5% application speedup. Ocean application with the contiguous memory allocation has better results compared to Radiosity mainly because of the simpler and more repetitive producer-consumer sharing patterns observed in that application. For the performance results, Barnes and FMM have again close values and their overall performance is limited by the ratio of potential consumers. In addition, they have a dynamic structure which makes it harder to pre-

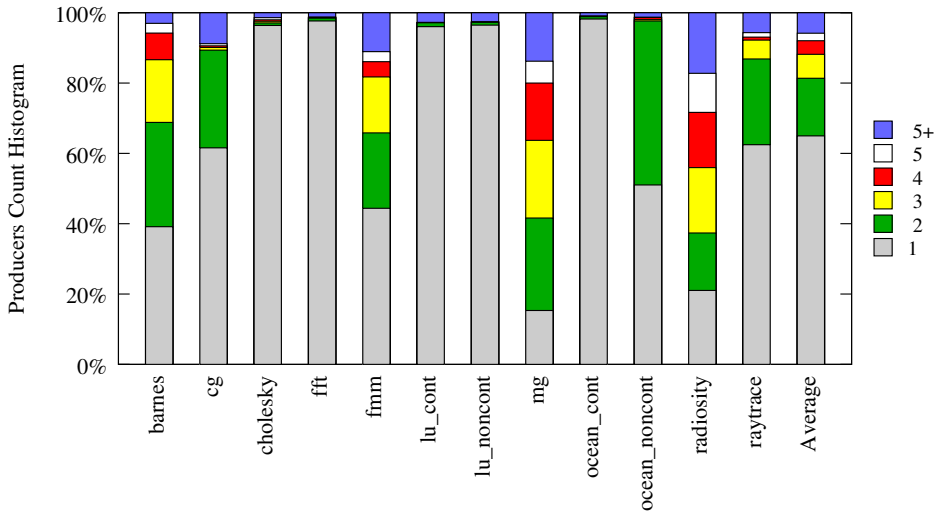


**Figure 4: Histogram of the number of producers for each memory block**

dict the consumer sets. The kernels from the NPB suite, MG and CG, have benefited from the adaptive cache coherence protocol as they include more than 10% of consumers. MG suffers from the multiple-producers effect compared to CG which resulted in lower coherence misses reduction. FFT and LU applications show closer improvements both for the coherence misses reduction and application speedup around 10%.

## 5.4 Sensitivity to PCPC size

In order to further analyze the performance of the adaptive cache coherence protocol, this section provides performance results for various PCPC sizes starting with 500 elements to 16K elements. For brevity, Figure 7 only shows results from MG and Cholesky applications which demonstrate two different behavior. MG performance is not affected until 8K elements and then see a big jump while going from 8K to 16K elements. On the other hand, Cholesky results exhibit less sensitivity to PCPC size. This result also illustrates the effectiveness of the Address-Filtering mechanism in PCPC. Furthermore, it also shows that coherence activities for many scientific applications are associated with a small fraction of the whole memory size that is being touched.

# 6. CONCLUSIONS AND FUTURE WORK

Cache coherence overhead has proven to be a performance bottleneck for shared-memory systems. As we see more and more cores put on a single chip, we will have CMPs with hundreds of cores in the near future with an underlying shared memory system. Accordingly, coherence misses will become even more important for upcoming CMPs as the cache-to-cache latencies will also increase. Contemporary shared-memory architectures employ write-invalidate based protocols which are known to generate coherence misses that yield to latency issues. On the other hand, write-update based protocols can solve the coherence misses problem but they tend to generate excessive network traffic which is especially not desirable for CMPs. Previous studies have shown that a single protocol approach is not sufficient for many sharing patterns. As a solution, we presented an adaptive cache coherence protocol which targets write-update optimizations for producer-consumer sharing patterns. This work targets a minimalistic hardware extension approach to test the benefits of such adaptive protocols in a practical environment.

Experimental study is conducted on a 16-core tiled CMP by using a full-system simulator with selected scientific applications from SPLASH-2 and NPB benchmarking suites. Results show up to 40% improvement for coherence misses which corresponds to almost 15% application speedup. Selected workloads were further profiled using trace-based simulation in order to interpret the results. This profiling analysis is also provided as a guide. In addition, we selected workloads with different sharing patterns and sizes to test the performance of our proposed work for broader set of applications. Although performance speedup numbers are highly sensitive to the implementation and underlying toolset restrictions, we show them to represent the overall effect on the execution time. We believe further performance gains can be obtained by more aggressive prediction mechanisms and also employing a weaker memory consistency model.

To test the cache coherence overhead and analyze potential remedies for cache-to-cache latencies for larger CMPs, we are working on 64-core and 128-core CMPs as an immediate extension to our study. Instruction-based predictors can be studied to evaluate the effectiveness compared to address-based predictions. In general, coherence predictor domain can be further investigated to tune the previous work for upcoming CMPs and many-core architectures.

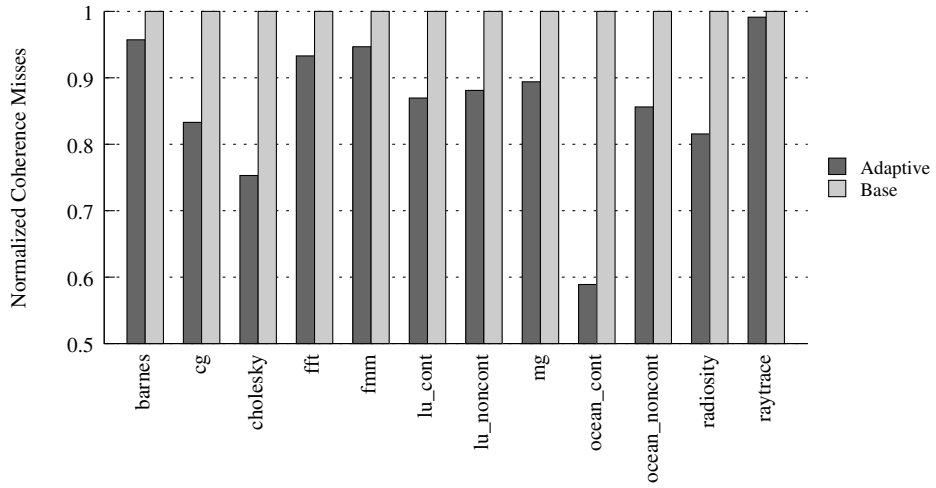


**Figure 5: Coherence Misses**

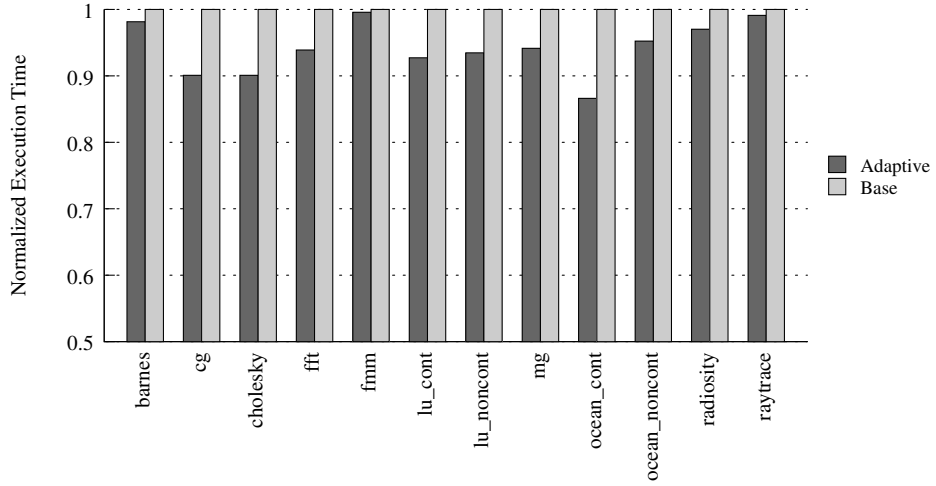


**Figure 6: Execution Time**

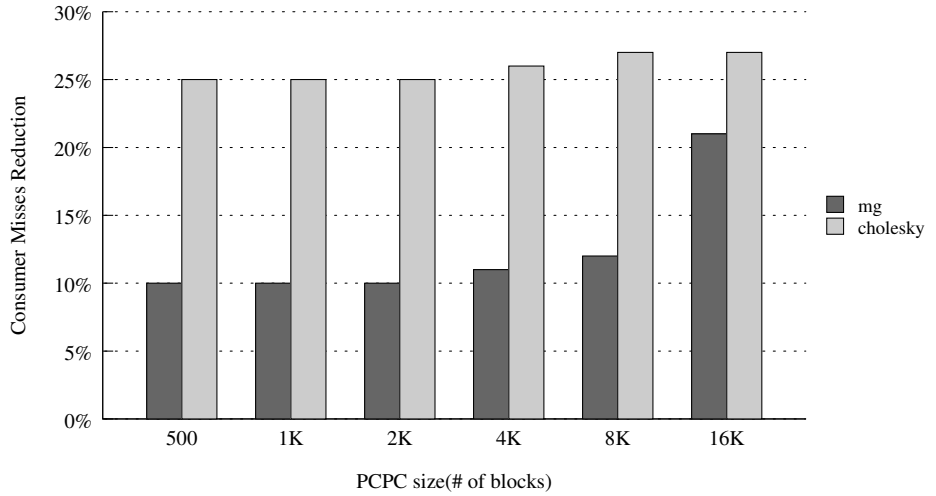


**Figure 7: Sensitivity to PCPC size based on obtained coherence misses reduction**

# 7. REFERENCES


[1] NAS Parallel Benchmarks, http://www.nas.nasa.gov/resources/software/npb.html.

[2] NAS Parallel Benchmarks, openmp version developed by omni group, http://www.hpcs.cs.tsukuba.ac.jp/omni-openmp.

[3] Teraflops research chip, http://techresearch.intel.com/articles/tera-scale/1449.htm.

[4] Tile-gx100, a 100-core microprocessor from Tilera corporation, http://www.tilera.com.

[5] H. Abdel-Shafi, J. Hall, S. V. Adve, and V. S. Adve. An evaluation of fine-grain producer-initiated communication in cache-coherent multiprocessors. In *International Symposium on High-Performance Computer Architecture (HPCA)*, pages 204–, 1997.

[6] M. Acacio, J. Gonzalez, J. Garcia, and J. Duato. A novel approach to reduce L2 miss latency in shared-memory multiprocessors. In *IPDPS '02: Proceedings of the International Parallel and Distributed Processing Symposium*, pages 62–69, 2002.

[7] M. Acacio, J. González, J. García, and J. Duato. Owner prediction for accelerating cache-to-cache transfer misses in a cc-NUMA architecture. In *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pages 1–12. IEEE Computer Society Press Los Alamitos, CA, USA, 2002.

[8] M. E. Acacio, J. González, J. M. García, and J. Duato. The use of prediction for accelerating upgrade misses in cc-NUMA multiprocessors. In *IEEE PACT*, pages 155–164. IEEE Computer Society, 2002.

[9] A. Agarwal and M. Levy. The kill rule for multicore. In *DAC '07: Proceedings of the 44th annual conference on Design automation*, pages 750–753. IEEE, 2007.

[10] S. R. Alam, R. F. Barrett, J. A. Kuehn, P. C. Roth, and J. S. Vetter. Characterization of scientific workloads on systems with multi-core processors. In *IISWC*, pages 225–236. IEEE, 2006.

[11] A. R. Alameldeen, M. M. K. Martin, C. J. Mauer, K. E. Moore, M. Xu, M. D. Hill, D. A. Wood, and D. J. Sorin. Simulating a $2m commercial server on a $2k pc. *IEEE Computer*, 36(2):50–57, 2003.

[12] C. Anderson and A. R. Karlin. Two adaptive hybrid cache coherency protocols. In *International Symposium on High-Performance Computer Architecture (HPCA)*, pages 303–313, 1996.

[13] S. Balakrishnan, R. Rajwar, M. Upton, and K. Lai. The impact of performance asymmetry in emerging multicore architectures. In *ISCA '05: Proceedings of the 32nd annual international symposium on Computer Architecture*, pages 506–517, Washington, DC, USA, 2005. IEEE Computer Society.

[14] E. E. Bilir, R. M. Dickson, Y. Hu, M. Plakal, D. J. Sorin, M. D. Hill, and D. A. Wood. Multicast snooping: A new coherence method using a multicast address network. In *ISCA '99: Proceedings of the 26th annual international symposium on Computer architecture*, pages 294–304, 1999.

[15] G. Byrd and M. Flynn. Producer-consumer communication in distributed shared memory multiprocessors. *Proceedings of the IEEE*, 87(3):456–466, Mar 1999.

[16] L. Cheng and J. B. Carter. Extending cc-numa systems to support write update optimizations. In *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, page 30. IEEE/ACM, 2008.

[17] L. Cheng, J. B. Carter, and D. Dai. An adaptive cache coherence protocol optimized for producer-consumer sharing. In *International Symposium on High-Performance Computer Architecture (HPCA)*, pages 328–339. IEEE Computer Society, 2007.

[18] L. Cheng, N. Muralimanohar, K. Ramani, R. Balasubramonian, and J. B. Carter. Interconnect-aware coherence protocols for chip multiprocessors. In *ISCA '06: Proceedings of the 33rd annual international symposium on Computer Architecture*, pages 339–351, Washington, DC, USA, 2006. IEEE Computer Society.

[19] M. Chu, R. Ravindran, and S. Mahlke. Data Access Partitioning for Fine-grain Parallelism on Multicore Architectures. In *MICRO '07: Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 369–380, Washington, DC, USA, 2007. IEEE Computer Society.

[20] A. L. Cox and R. J. Fowler. Adaptive cache coherency for detecting migratory shared data. In *International Symposium on Computer Architecture (ISCA)*, pages 98–108, 1993.

[21] F. Dahlgren. Boosting the performance of hybrid snooping cache protocols. In *ISCA '95: Proceedings of the 22nd annual international symposium on Computer architecture*, pages 60–69, New York, NY, USA, 1995. ACM.

[22] F. Dahlgren and P. Stenström. Reducing the write traffic for a hybrid cache protocol. In *International Conference on Parallel Processing (ICPP)*, pages 166–173, 1994.

[23] N. Eisley, L.-S. Peh, and L. Shang. In-network cache coherence. In *MICRO 39: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 321–332, Washington, DC, USA, 2006. IEEE Computer Society.

[24] D. Geer. Industry trends: Chip makers turn to multicore processors. *IEEE Computer*, 38(5):11–13, 2005.

[25] D. Ghosh, J. B. Carter, and H. D. III. Perceptron-based coherence predictors. In *Proc. of 2nd Workshop on Chip Multiprocessor Memory Systems and Interconnects (CMP-MSI), in conjunction with ISCA 2008.*

[26] P. F. Gorder. Multicore processors for science and engineering. *Computing in Science and Eng.*, 9(2):3–7, 2007.

[27] H. K. Grahn and P. Stenström. Evaluation of a competitive-update cache coherence protocol with migratory data detection. *Journal of Parallel and Distributed Computing*, 39:39–2, 1996.

[28] D. Gustavson. The scalable coherent interface and related standards projects. *Micro, IEEE*, 12(1):10–22, Feb 1992.

[29] A. R. Karlin, M. S. Manasse, L. Rudolph, and D. D. Sleator. Competitive snoopy caching. *Algorithmica*, 3:77–119, 1988.

[30] S. Kaxiras and J. R. Goodman. Improving cc-NUMA

performance using instruction-based prediction. In *International Symposium on High-Performance Computer Architecture (HPCA)*, pages 161–, 1999.

[31] S. Kaxiras and C. Young. Coherence communication prediction in shared-memory multiprocessors. In *International Symposium on High-Performance Computer Architecture (HPCA)*, pages 156–167, 2000.

[32] A. Kayi, E. Kornkven, T. El-Ghazawi, S. Al-Bahra, and G. B. Newby. Performance analysis and tuning for clusters with ccnuma nodes for scientific computing - a case study. *International Journal of Computer Systems Science and Engineering*, 24(5), September 2009.

[33] D. Koufaty, X. Chen, D. K. Poulsen, and J. Torrellas. Data forwarding in scalable shared-memory multiprocessors. In *International Conference on Supercomputing (ICS)*, pages 255–264, 1995.

[34] A.-C. Lai and B. Falsafi. Memory sharing predictor: The key to a speculative coherent dsm. In *ISCA '99: Proceedings of the 26th annual international symposium on Computer architecture*, pages 172–183, 1999.

[35] A.-C. Lai and B. Falsafi. Selective, accurate, and timely self-invalidation using last-touch prediction. In *International Symposium on Computer Architecture (ISCA)*, pages 139–148, 2000.

[36] A. R. Lebeck and D. A. Wood. Dynamic self-invalidation: Reducing coherence overhead in shared-memory multiprocessors. In *ISCA '95: Proceedings of the 22nd annual international symposium on Computer architecture*, pages 48–59, 1995.

[37] S. Leventhal and M. Franklin. Perceptron based consumer prediction in shared-memory multiprocessors. In *ICCD 2006: International Conference on Computer Design*, pages 148–154, Oct. 2006.

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

[39] M. M. K. Martin. Formal verification and its impact on the snooping versus directory protocol debate. In *ICCD 2005: International Conference on Computer Design*, pages 543–449. IEEE Computer Society, 2005.

[40] M. M. K. Martin, P. J. Harper, D. J. Sorin, M. D. Hill, and D. A. Wood. Using destination-set prediction to improve the latency/bandwidth tradeoff in shared-memory multiprocessors. In *International Symposium on Computer Architecture (ISCA)*, pages 206–217. IEEE Computer Society, 2003.

[41] M. M. K. Martin, D. J. Sorin, A. Ailamaki, A. R. Alameldeen, R. M. Dickson, C. J. Mauer, K. E. Moore, M. Plakal, M. D. Hill, and D. A. Wood. Timestamp snooping: an approach for extending smps. In *International conference on Architectural support for programming languages and operating systems (ASPLOS)*, pages 25–36, 2000.

[42] M. M. K. Martin, D. J. Sorin, M. D. Hill, and D. A. Wood. Bandwidth adaptive snooping. In *International Symposium on High-Performance Computer Architecture (HPCA)*, pages 251–262, 2002.

[43] M. R. Marty, J. D. Bingham, M. D. Hill, A. J. Hu, M. M. K. Martin, and D. A. Wood. Improving multiple-cmp systems using token coherence. In *ISCA '05: Proceedings of the 32nd annual international symposium on Computer Architecture*, pages 328–339. IEEE Computer Society, 2005.

[44] S. S. Mukherjee and M. D. Hill. Using prediction to accelerate coherence protocols. In *International Symposium on Computer Architecture (ISCA)*, pages 179–190, 1998.

[45] H. Nilsson and P. Stenström. An adaptive update-based cache coherence protocol for reduction of miss rate and traffic. In *Proc. Parallel Architectures and Languages Europe (PARLE) Conf., Athens, Greece (Lecture Notes in Computer Science, 817*, pages 363–374. Springer-Verlag, 1994.

[46] J. Nilsson and F. Dahlgren. Improving performance of load-store sequences for transaction processing workloads on multiprocessors. In *International Conference on Parallel Processing (ICPP)*, pages 246–, 1999.

[47] J. Nilsson and F. Dahlgren. Reducing ownership overhead for Load-Store sequences in cache-coherent multiprocessors. In *IPDPS '00: Proceedings of the International Parallel and Distributed Processing Symposium*, pages 684–692. IEEE Computer Society, 2000.

[48] J. Nilsson, A. Landin, and P. Stenström. The coherence predictor cache: A resource-efficient and accurate coherence prediction infrastructure. In *IPDPS '03: Proceedings of the International Parallel and Distributed Processing Symposium*, page 10. IEEE Computer Society, 2003.

[49] K. Olukotun and L. Hammond. The future of microprocessors. *Queue*, 3(7):26–29, 2005.

[50] A. Raynaud, Z. Zhang, and J. Torrellas. Distance-adaptive update protocols for scalable shared-memory multiprocessors. In *HPCA '96: Proceedings of the Second International Symposium on High-Performance Computer Architecture*, pages 323–334, Feb 1996.

[51] P. Stenström, M. Brorsson, and L. Sandberg. An adaptive cache coherence protocol optimized for migratory sharing. In *International Symposium on Computer Architecture (ISCA)*, pages 109–118, 1993.

[52] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 Programs: Characterization and methodological considerations. In *ISCA '95: Proceedings of the 22nd annual international symposium on Computer architecture*, pages 24–36, 1995.

[53] T.-Y. Yeh and Y. N. Patt. Alternative implementations of two-level adaptive branch prediction. In *International Symposium on Computer Architecture (ISCA)*, pages 124–134, 1992.