Performance Analysis of Cache Coherence Protocols for Multi-core Architectures : A System Attribute Perspective

Amit D. Joshi
Department of Computer Science and Engineering
National Institute of Technology, Tiruchirappalli
adj.comp@coep.ac.in

Satyanarayana Vollala
Department of Computer Science and Engineering
National Institute of Technology, Tiruchirappalli
satya4nitt@gmail.com

B. Shameedha Begum
Department of Computer Science and Engineering
National Institute of Technology, Tiruchirappalli
shameedha@nitt.edu

N. Ramasubramanian
Department of Computer Science and Engineering
National Institute of Technology, Tiruchirappalli
nrs@nitt.edu

ABSTRACT

Shared memory multi-core processors are becoming dominant in todays computer architectures. Caching of shared data may produce a problem of replication in multiple caches. Replication provides reduction in contention for shared data items along with reduction in access latency and memory bandwidth. Caching of shared data that are being read by multiple processors simultaneously, introduces the problem of cache coherence. There are two different techniques to track the sharing status viz. Directory and Snooping. This work gives an emphasis on the study and analysis of impact of various system parameters on the performance of the basic techniques. The performance analysis of this work is based on the number of processors, available bandwidth and cache size. The prime aim of this work is to identify appropriate cache coherence protocol for various configurations. Simulation results have shown that snooping based systems are appropriate for high bandwidth systems and is the ideal choice for CPU and communication intensive workloads while directory based cache coherence protocols are suitable for lower bandwidth systems and will be more appropriate for memory intensive workloads.

CCS Concepts

•Computer systems organization → Multicore architectures;

Keywords

Multi-core; Cache Coherence; Directory; Snooping; Eviction; Clustering; AMAT.

1. INTRODUCTION

Unicore processor has a single central processing unit on a single chip. Earlier these processors were large and had slow computation speed. With the advent of newer technologies and higher level of integration, the single chip was fabricated with single central processing unit along with a cache memory. There may also be multiple levels of cache memory on a single chip. The performance improvement has dropped to 20% due to power dissipation of chips, little instruction level parallelism is availed without change in memory latency. And hence, these problems lead to the integration of more than one core on a single die [1].

Multiple processors per chip gives a way to achieve high performance. A multiple processor is having different types like dual core, quad core, hexa core. Each of them has two cores, four cores and six cores respectively [2]. Reduction in the memory bandwidth demands of a processor is fulfilled by large multi level caches. Reduction in single processor′s main memory bandwidth demand will turn multiple processors to share the same memory. Many designers created small cache multiprocessors which shared a single physical memory with a shared bus. Early designs of multi-processors were with a processor and a cache subsystem on a board which connected into the bus back plane. Subsequent versions designed with two to four processors per board to achieve higher densities which used multiple busses with interleaved memory [3]. The small sized processors along with reduced bus bandwidth achieved by large caches are symmetric multi-processors with cost effectiveness and sufficient memory bandwidth.

Caching of shared and private data is usually supported by the symmetric shared memory processor. Shared values may be replicated in multiple caches. Replication reduces the contention exists for shared data read by a multiple processors simultaneously. Replication also reduces access latency and memory bandwidth. Shared data caching introduces a problem of cache coherence [4]. Tracking the state of any shared data block is a key to implement cache coherence. In a snooping protocol, the write invalidate ensures that whether a processor has right to access a data item before writing it. Invalidations are performed on other copies on a write. The write update protocol updates all cached copies on a write. Consumption of more bandwidth is observed in a write update due to broadcast of all write to shared cache lines [5].

Directory protocol reduces the bandwidth demands in centralized shared memory systems. The state of every block is maintained by a directory. A directory protocol design is

AICTC '16, August 12-13, 2016, Bikaner, India

DOI: http://dx.doi.org/10.1145/2979779.2979801

applicable to both, a distributed memory and a centralized memory organized into banks [6]. A directory may become a bottleneck. To avoid this the directory is distributed so that different accesses can reach to different directories. Keeping a sharing status of a block always in a single known location directory protocol avoids broadcast [3]. We observe that increase in cache size increases the performance of snoopy protocols more than for directory protocols [7]. A commonly used protocol is the MESI protocol. Its supports write back cache. Intel has used this protocol in the Pentium processor to support the efficient right back cache in addition to the write through cache that was in Intel 486 processor [8] [9]. Every cache line is marked either of the four states namely Modified, Exclusive, Shared and Invalid.

MOESI protocol has all possible states used by other protocols. In addition to the states in MESI protocol, there is an Owned state that represents data with modified and shared state. The need of writing modified data back to main memory is avoided due to this before sharing it. Eventual writing of data using write back may be deferred. Direct cache to cache data transfer must be possible so that modified state data can be provided to yet another reader without transferring it to memory.

2. LITERATURE REVIEW

A Snoopy protocol is depending on a common bus monitored by all caches. The nature of the bus gives a solution to cache coherence. Each wire of a bus can observe every bus transaction. A processor request to its cache is a checking done by the cache controller for its state. After checking the cache state required action is taken. The snooping on bus and monitoring the transactions by all cache controllers maintains coherence.

All cache controllers observe all the transactions on a bus. The transactions are visible in the same order. Every write is observed by every cache controller. Snooping works either by invalidating or updating its copy. So snoopy protocols are classified into two types viz. invalidation based protocol and update based protocol [10] [11]. Whenever a processor with the copy accesses the block will get the recently modified value. The broadcast nature of snoopy protocol makes it more bandwidth consuming. Due to more bandwidth consumption the snoopy protocol obtains data quickly. It is limited to small scale systems [12].

A state bit in a directory protocol provides the state of block. The state may be either uncached, shared or exclusive. The state and presents bits give an information about the caches to be invalidated. Read miss identifies the different copies of the block. An invalidate or update may received by these copies. Directories keep track of all the blocks that are present in different nodes. This eliminates the need of a broadcast. On a read miss, the block will be either in main memory or it will be as an exclusive copy. Directory will tell the location of this exclusive copy. The broadcast approach is better for a write miss if the sharers are small in number and not increasing with number of nodes. The directory based system is the scalable solution for a greater number of nodes. The indirection overhead associated with directory protocol degrades its performance than Snoopy protocol [13].

High performance can be achieved by using efficient and scalable protocols like directory protocol. Limited directory may give evictions, invalidations and hence the performance is get degraded. The protocol deactivation gives a way for omitting tracking of blocks. This reduces the load and increases the effective size [14].

Proximity-Aware Directory based Coherence : This scheme initiates cache to cache transfers. Due to this read and write misses get accelerated. The reduction in network traffic along with off chip memory accesses are the benefits given by this scheme [15].

Improving Memory Utilization : The dynamically tagged directory system allocates pointers to all blocks. It is necessary to show that the software hardware combined approach is significant along with less memory requirement. Simulations proves that it is less sensitive with respect to memory disambiguation and inter procedural analysis [6].

Enhance Cache Communications : Due to unavailability of explicit communication support, the inter core communications in multi-core processors depend on coherence protocols. This requires demand based cache line transfer. There is a need of improving performance significantly along with reduction in L3 cache misses as mentioned in [16].

Shrinking Coherence Directory : Coherence directory is a challenge with respect to scalability in multi-core chips. This scheme uses SPATL: Sharing Pattern based Tagless Directory. The sharing pattern commonality is exploited by SPATL. The sharing patterns and bloom filters get decoupled by SPATL. Elimination of redundant copies of sharing patterns is done with SPATL. SPATL works with inclusive and non-inclusive shared caches. It gives significant storage saving over the taggless directory with 16-cores. Scalability of SPATL is better than idealized directory. It is scalable for 1024 chips with very less private cache space as described in [17].

3. SCOPE AND OBJECTIVES

The aim of this work is to study and analyze the impact of various system parameters on the performance of the two basic schemes viz. Snooping and Directory. The parameters analyzed include the number of processors in the multiprocessor system, the bandwidth available on the interconnection network, and the processor cache size. The prime aim of this work is to identify appropriate cache coherence protocol for the given configuration along with the available resources. System performance is correlated with Average Memory Access Time (AMAT) of systemś memory hierarchy. AMAT can be formulated and can correlated with system performance as shown in the following expressions :

AMAT = Hit time + Miss rate × Miss penalty

$Hit\,time_{L1} + Miss\,Rate_{L1}$ $(Hit\,time_{Li} + Miss\,rate_{Li} \times Miss\,penalty_{Li})$, Where i can be any integral number depending on the number of cache levels.

CPU execution time = (CPU execution cycles + Memory stall cycles) ×Clock cycle time

4. IMPLEMENTATION OF PROTOCOLS

The different cache coherence protocols viz. MI, MESI, MOESI have been implemented for two cache coherence classes namely Snooping and Directory. ALPHA ISA has been built for all the implemented cache coherence protocols. PARSEC benchmarks are used to run over these simulated systems. Results are compared to visualize the

performance of two different classes of the cache coherence protocol. ARM ISA has been built for MI based Snooping protocol. The more realistic benchmark has been run over the simulated ARM system.

4.1 Experimental Setup

We used the x86 system with a configuration as shown in Table 1. The gem5 simulator has been used with ALPHA system setup in it. The analysis of results of different cache coherence protocols are observed by simulating ALPHA system with gem5 simulator. The results analyzed are with different system parameters of multi-core architectures. The benchmarks used in this simulation include PARASEC and BBENCH for ALPHA system and ARM system respectively. We used different CPU models for ALPHA and ARM systems. The models uses inorder and out of order executions respectively. The basic configuration of the system is given in Table 2.

4.2 PARSEC Experiment

All experiments are run on the target platform gem5 simulated ALPHA model with different number of cores ranging from 1 to 8. The PARSEC benchmarks used include Blackscholes, Bodytrack, Canneal, Fluidanimete, Swaption, Streamcluster, Vips, x264.

As PARSEC is multi-threaded, it is important to understand the communication patterns between CPUs in each benchmark, as each benchmark may exhibit different sharing or consumer-producer patterns. The PARSEC benchmark suite offers six standardized input sets. According to [18] [19], the properties of both the larger and smaller input sets are same. As such, to enable exploration of a large set of configurations, the simsmall input set was used. According to [20] the PARSEC benchmark set is diverse in the synchronization methods in its benchmarks use. This poses PARSEC as an appropriate representative workload bundle described in [21] [22].

4.3 BBENCH Experiment

In order to perform full system performance analysis, it is crucial to make use of real world workloads, such that will stress the entire system, rather than mostly the CPUs. Smartphones, web browsers are the most representative applications. Browser bench is an interactive smartphone benchmark suite that includes a web-browser page rendering benchmark. BBench is automated, repeatable, and fully contained as described in [23]. It exercises not only the web-browser, but the underlying libraries and operating system. BBench is multi-threaded, comparing to the SPEC benchmark suite which is single threaded. It has a much higher mispredictions rate due to code size and sparseness. BBench makes use of more shared libraries and system calls. Its core is the rendering of ten of the most frequently used and complex sites on the website.

5. RESULTS AND DISCUSSIONS

5.1 Performance Analysis

5.1.1 PARSEC Results

The results provided in the following section are based on the outcome of the simulation made with PARSEC benchmarks. AMAT is influenced by hit time, miss rate and miss

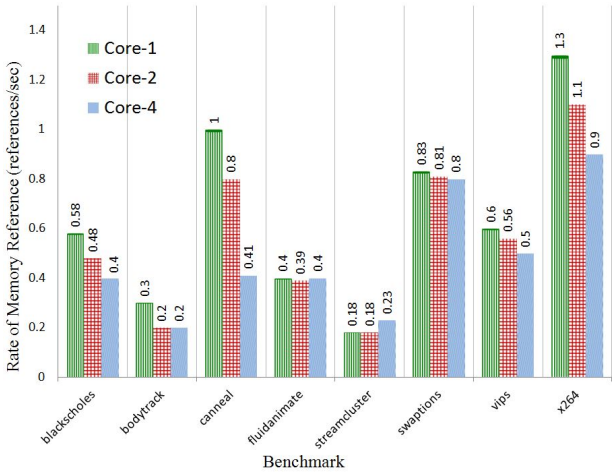

Figure 1: Rate of memory references of PARSEC benchmarks

latency. This AMAT is correlated with the execution time of the system and hence performance of the system.

Benchmark Behavior : The rate of memory references as shown in Figure 1 shows how memory intensive each benchmark is, and how its dependence on the memory system scales as more CPUs are added. There are several different patterns, for instance, for blackscholes we can observe a linear decrease as a function of the CPUs in the system. The swaptions benchmark, on the other hand, exhibits a fairly constant rate of memory references. In stream cluster, adding the core significantly increases the number of memory references. x264 is highly memory intensive benchmark.

Bus Traffic : Figure 2 shows the variation of bus traffic for Snooping and Directory based cache coherence protocol with different number of cores on gem5 simulated ALPHA ISA system with the configuration depicted in Table 2. In uniprocessor directory based system has higher bus traffic irrespective of snooping based system. In dual core, both the protocols experience similar kind of traffic. Beyond dual core, directory based system has lesser bus traffic than snooping based system. There is no much variation between the two protocols from two cores to four cores. As the number of cores increases, each processor shares the common bus and hence number of coherent transactions also increases. This leads to congestion on the shared bus and hence directory based cache coherence protocol performs better than snooping cache coherence protocol.

Miss Latency : Figure 3 shows the variation of miss latency in cycles in different directory based cache coherence protocol with different number of cores on gem5 simulated ALPHA ISA system. MI directory based cache coherence protocol for uniprocessor shows high miss latency value hence it is excluded for further simulations. MESI and MOESI shows almost same miss latency values independent of the number of cores.

L1 Invalidate Message Traffic : Figure 4 shows invalidate message traffic in directory based coherent system. As core count increases, number of invalidate messages are increasing since number of shared blocks also increases. This leads to more coherence misses. Further, graph shows lower

Table 1: Base Hardware Configuration

ISA	Coherence Protocol	Number of Cores	L1 Size		L1 Assoc		LLC Size	LLC Assoc	Replacement Policy
			IC	DC	IC	DC			
x86	MOESI_CMP	2	256KB	256KB	8	8	1MB	16	PLRU

Table 2: Configuration of Baseline System

ISA	Coherence Protocol	Core count	L1 Size		L1 Assoc		LLC Size	LLC Assoc
			IC	DC	IC	DC		
ALPHA	MI, MESI & MOESI	1-8	32KB	32KB	8	8	8MB	16
ARM	MESI	2	32KB	32KB	8	8	8MB	16

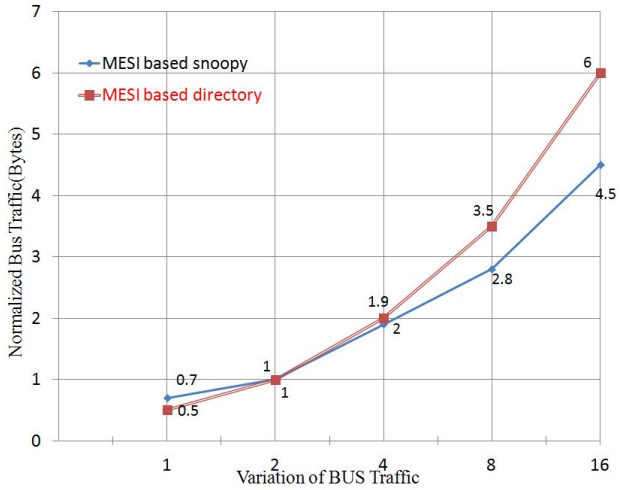

Figure 2: Variation of Bus Traffic

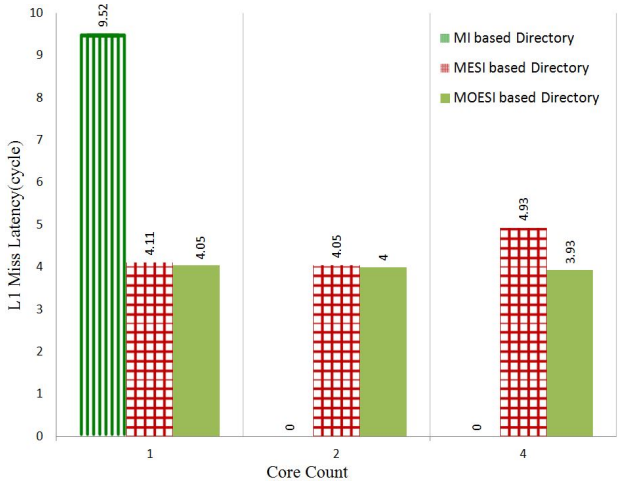

Figure 3: Variation of Miss Latency

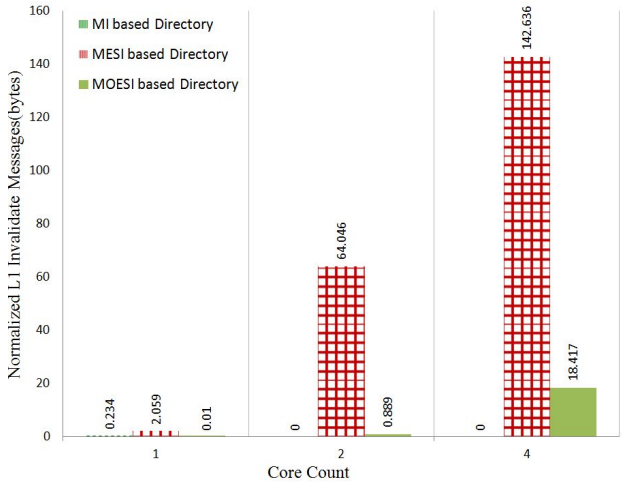

Figure 4: Invalidate Message Traffic

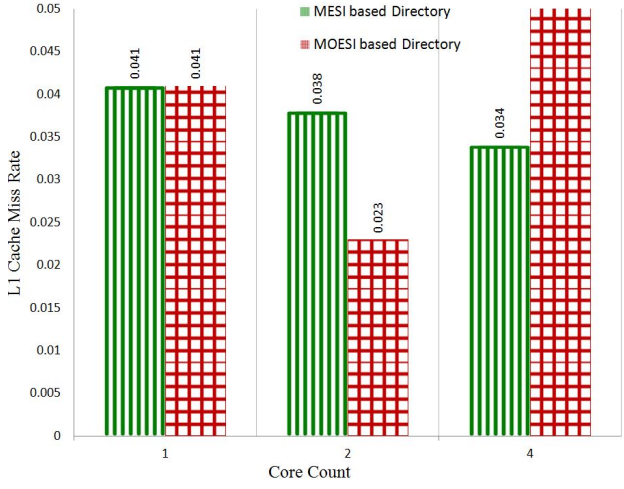

Figure 5: Variation of L1 Cache Miss Rate of MESI and MOESI

L1 invalidate message traffic for MOESI with different number of cores due to the presence of one more transition state leading to lower number of invalid blocks.

Figure 5 shows the variation of of L1 cache miss rate of MESI and MOESI directory based cache coherence protocol with different number of cores on gem5 simulated ALPHA ISA system. MOESI shows lower miss rate than MESI. This is due to lower number of invalid blocks in L1 cache in MOESI.

L1 cache miss rate : Figure 6 shows the variation of L1 cache miss rate of snooping and directory based cache

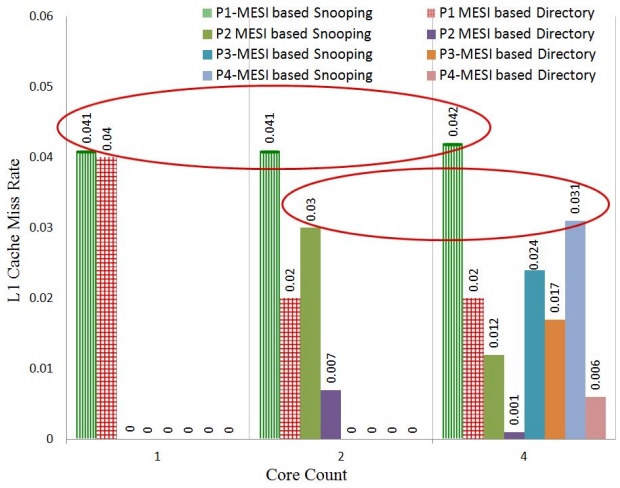

Figure 6: Variation of L1 Cache Miss Rate for Snooping and Directory vs Core Count

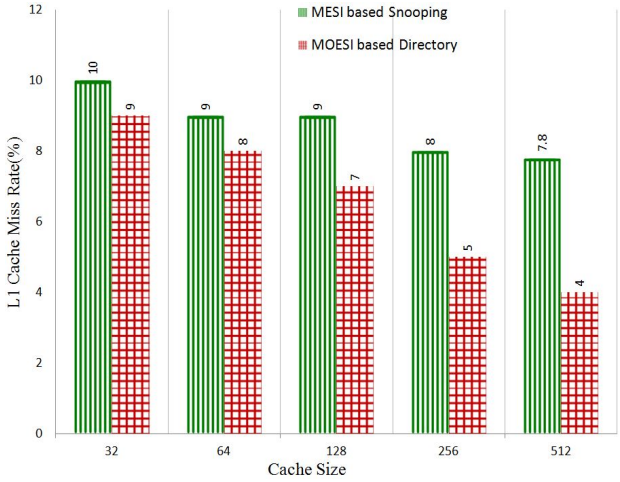

Figure 7: Variation of L1 Cache Miss Rate of Snooping and Directory vs Cache Size

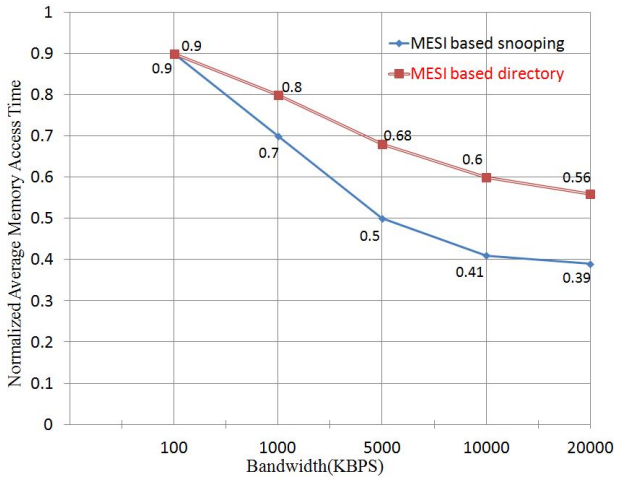

Figure 8: Variation of L1 Cache Miss Rate of Snooping and Directory vs Cache Size

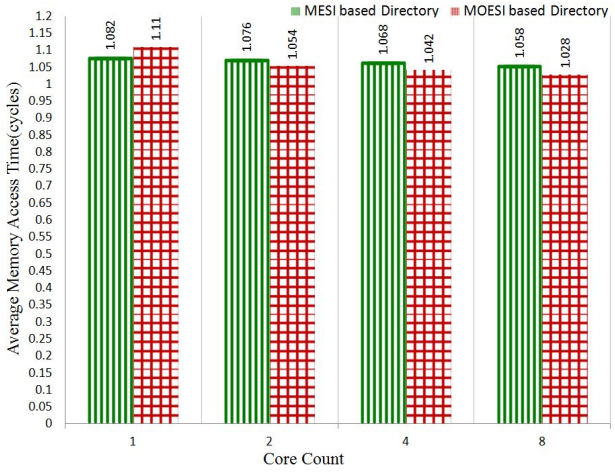

Figure 9: Variation of AMAT vs Core Count

coherence protocol with different number of cores on gem5 simulated ALPHA ISA system. Miss rate is shown for each core in multi-core system. L1 cache miss rate is almost same for core 0 and core 1. The decrease in miss rate with respect to more number of cores is not observed. This is due to the coherence misses get increased with core count. Average miss rate for different core count shows linear decrease since conflict miss decreases with little increase in coherence miss with the increase in core count.

Figure 7 shows variation of L1 cache miss rate of snooping and directory based cache coherence protocol with increase in L1 cache size on gem5 simulated ALPHA ISA system. There is significant decrease in miss rate with increase in the size of L1 cache since there is decrease in capacity miss without increase in coherence miss.

AMAT vs Bandwidth : Figure 8 shows the variation of AMAT of snooping and directory based cache coherence protocol with increase in bandwidth on gem5 simulated ALPHA ISA system. Beyond 100kbps bandwidth, snooping has better AMAT than directory based cache coherence protocol since large number of data blocks can be transferred on the high bandwidth bus without much congestion. AMAT gets saturated at 20 Mbps.

AMAT vs Core Count : Figure 9 shows the variation of AMAT of snooping and directory based cache coherence protocol with different core count simulated ALPHA ISA system. For uniprocessor, miss penalty of directory based dominates resulting in higher AMAT since miss penalty of directory dominates over miss rate and hit rate of directory protocol. With the increase in the number of cores, AMAT decreases linearly with small value since core count has not a significant effect on miss rate. The octa core configuration gives a difference in AMAT with respect to snoopy and directory protocols. This is due to high coherence traffic in snoopy protocol.

5.1.2 BBENCH Results

The results demonstrate that a significant impact from delaying snoop responses in a gem5 like system when high levels of inter-cluster sharing is expected. Currently even in PARSEC, which is a typical benchmark for such relatively

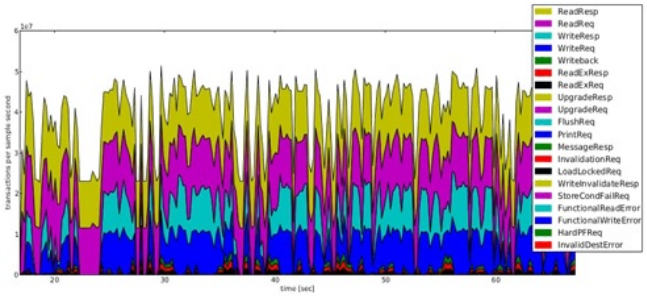

Figure 10: Variation of AMAT vs Core Count

Table 3: Summary of Simulation Results WRT Core Count

Parameter	Core 1	Core 2	Core 4
Bus Traffic	Snoopy better	Both Same	Dir better
Invalidate Messages	MESI and MOESI same	MOESI better	MOESI better
Miss Latency	MI worst latency	MOESI and MESI same	MOESI and MESI same

intensive multi threaded workloads, this impact is not critical and in many cases hardly noticeable.

The results shown in Figure 10 cover the less interesting systems, those which only contain one cluster. The experiments with multi-clustered systems should be rerun once the open bug in gem5ś caches is fixed, to evaluate the impact in such systems.

The bus snoop layer statistics, notably the utilization and aggregate data statistics, can determine how much inter-cluster sharing there actually exists, as they represent snoop-hits from two remote clusters.

5.2 Inference from Results

From the Table 3 to Table 6 show the simulation results with respect to core count, L1 cache miss rate, AMAT respectively. Table 4 shows the appropriate cache coherence protocol for different classes of workloads. As the core count increases bandwidth demand increases in snoopy protocol. The increased traffic in snoopy protocol put a limitation on the scalability irrespective of directory protocol. MOESI outcomes its predecessor MESI cache coherence protocol in having lesser L1 invalidate message traffic. This leads to lower cache miss rate in MOESI when compared with MESI cache coherence protocol. Increase in L1 cache size has more impact than increase in number of cores to reduce cache miss rate for coherent system. Directory based cache coherence protocol outcomes snoop based cache coherence protocol at lower bandwidth. Snooping outcomes the directory based cache coherence protocol at higher bandwidth system. Snooping has better AMAT for uniprocessor systems but directory has better AMAT for multi-core systems. MOESI Directory based cache coherence protocol seems to give better system performance for multi-core platforms that works under lower bandwidths.

6. CONCLUSION

MOESI Directory based cache coherence protocol shows higher performance for multicores than MI, MESI Directory and Snooping based cache coherence protocol. Clustering of cores corresponding to interleaved shared cache memory increases the performance of directory based multicore systems. For high bandwidth system, Snooping shows commendable result due to lesser overhead and reduced congestion. Directory based system does not fit for this class of systems. For low bandwidth system, Directory outcomes snooping based cache coherence protocol due to lesser congestion problem. Snooping based cache coherence protocol is apt for communication intensive and CPU intensive workloads. Multicasting irrespective of broadcasting of messages in case of snooping based cache coherence protocol will reduce the coherence traffic and can increase the scalability of the system. System with separate bus for coherence transactions in case of snooping based cache coherence protocol increase till definitely increase the scalability of the system. Reduction in the number of bits storing the information of processors that shares the same data block can reduce the overhead of the directory protocol. Filteration of cache blocks stored in the directory cache to keep track of true shared blocks will reduce the number of evictions in the directory cache and hence resulting in reduce evictions from cache memory. For the systems with varying bandwidth , wireless and adhoc networks, hybrid cache coherence protocol can be adopted that can make use of directory based for lower bandwidth networks and snoopy based for higher bandwidth.

Table 4: Summary of Simulation Results w.r.t. Cache Miss rate

Parameter	L1 Cache Miss Rate
Core 1	MESI and MOESI both same
Core 2	MOESI is better than MESI
Core 4	MOESI is better than MESI
L1 Cache Size (Multicore)	Dir better than Snoop, miss rate reduces with increase in size

Table 5: Summary of Simulation Results w.r.t. AMAT

Parameter	AMAT
Core 1	Snoop better than Dir
Core 2	Dir better than Snoop
Core 4	Dir outcomes Snoop
Low Bandwidth (<100Kbps, multicore)	Dir better than Snoop
High Bandwidth	Snoop better than Dir

7. REFERENCES

[1] David Geer. Chip makers turn to multicore processors. *Computer*, 38(5):11–13, 2005.

[2] Amit Golander, Nadav Levison, Omer Heymann, Alexander Briskman, Mark J Wolski, and Eric F Robinson. A cost-efficient l1–l2 multicore interconnect: Performance, power, and area considerations. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 58(3):529–538, 2011.

[3] John L Hennessy and David A Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2011.

Table 6: Summary of Simulation Results WRT Worklaods(Multicore)

Workload	Snooping	Directory
CPU Intensive	✓	
Memory Intensive		✓
Communication Intensive	✓	

[4] Daniel J Sorin, Mark D Hill, and David A Wood. A primer on memory consistency and cache coherence. *Synthesis Lectures on Computer Architecture*, 6(3):1–212, 2011.

[5] Muge Guher. Physical design of snoop-based cache coherence on multiprocessors.

[6] David J Lilja and Pen-Chung Yew. Improving memory utilization in cache coherence directories. *Parallel and Distributed Systems, IEEE Transactions on*, 4(10):1130–1146, 1993.

[7] David Chaiken, Craig Fields, Kiyoshi Kurihara, and Anant Agarwal. Directory-based cache coherence in large-scale multiprocessors. *Computer*, 23(6):49–58, 1990.

[8] Mark S Papamarcos and Janak H Patel. A low-overhead coherence solution for multiprocessors with private cache memories. In *ACM SIGARCH Computer Architecture News*, volume 12, pages 348–354. ACM, 1984.

[9] Per Stenström. A survey of cache coherence schemes for multiprocessors. *Computer*, 23(6):12–24, 1990.

[10] Lucien M Censier and Paul Feautrier. A new solution to coherence problems in multicache systems. *Computers, IEEE Transactions on*, 100(12):1112–1118, 1978.

[11] Michael R Marty and Mark D Hill. Virtual hierarchies to support server consolidation. In *ACM SIGARCH Computer Architecture News*, volume 35, pages 46–56. ACM, 2007.

[12] Michael R Marty. *Cache coherence techniques for multicore processors*. ProQuest, 2008.

[13] Rajeev Balasubramonian, Norman P Jouppi, and Naveen Muralimanohar. Multi-core cache hierarchies. *Synthesis Lectures on Computer Architecture*, 6(3):1–153, 2011.

[14] Blas A Cuesta, Alberto Ros, María E Gómez, Antonio Robles, and José F Duato. Increasing the effectiveness of directory caches by deactivating coherence for private memory blocks. In *ACM SIGARCH Computer Architecture News*, volume 39, pages 93–104. ACM, 2011.

[15] Jeffery A Brown, Rakesh Kumar, and Dean Tullsen. Proximity-aware directory-based coherence for multi-core processor architectures. In *Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures*, pages 126–134. ACM, 2007.

[16] Sevin Fide and Stephen Jenks. Proactive use of shared l3 caches to enhance cache communications in multi-core processors. *Computer Architecture Letters*, 7(2):57–60, 2008.

[17] Hongzhou Zhao, Arrvindh Shriraman, Sandhya Dwarkadas, and Vijayalakshmi Srinivasan. Spatl: Honey, i shrunk the coherence directory. In *Parallel Architectures and Compilation Techniques (PACT), 2011 International Conference on*, pages 33–44. IEEE, 2011.

[18] Nick Barrow-Williams, Christian Fensch, and Simon Moore. A communication characterisation of splash-2 and parsec. In *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, pages 86–97. IEEE, 2009.

[19] Thomas B Berg. Maintaining i/o data coherence in embedded multicore systems. *IEEE micro*, (3):10–19, 2009.

[20] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. The parsec benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 72–81. ACM, 2008.

[21] Christian Bienia and Kai Li. Fidelity and scaling of the parsec benchmark inputs. In *Workload Characterization (IISWC), 2010 IEEE International Symposium on*, pages 1–10. IEEE, 2010.

[22] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta. The splash-2 programs: Characterization and methodological considerations. In *ACM SIGARCH computer architecture news*, volume 23, pages 24–36. ACM, 1995.

[23] Anthony Gutierrez, Ronald G Dreslinski, Thomas F Wenisch, Trevor Mudge, Ali Saidi, Chris Emmons, and Nigel Paver. Full-system analysis and characterization of interactive smartphone applications. In *Workload Characterization (IISWC), 2011 IEEE International Symposium on*, pages 81–90. IEEE, 2011.