Exploiting Reuse-Frequency with Speculative and Dynamic Updates in an Enhanced Directory Based Coherence Protocol

Nilufar Ferdous¹, Monobrata Deb Nath¹, Byeong Kil Lee², and Eugene John¹
¹ Department of Electrical and Computer Engineering, The University of Texas at San Antonio, San Antonio, TX-78249, USA
² Samsung Research, Seoul, South Korea

Abstract - As the number of cores increases on chip multiprocessors, cache coherence is fast becoming a major impediment in improving the performance of the multi-cores. This is exacerbated by the fact that the interconnection speed does not scale well enough with the speed of the processors. To ameliorate these limitations, several mechanisms were augmented to the cache coherence protocols to enhance the performance of the multiprocessors. These mechanisms include policies such as write-update policy, write-invalidate policy etc. However, it has been previously shown that pure write-update protocol is highly undesirable because of the heavy traffic caused by the updates. On the other hand, write-invalidation protocol is not the optimal solution as many of the sharers of the cache blocks may be reused in the near future. In order to increase the efficiency, we introduce a novel update mechanism which uses reuse frequency and last touch time of each cache block as metrics to take the decision dynamically whether it will retain its write-invalidate strategy or will update the sharers as in write-update protocol. Our research enhances the speedup of MOESI cache coherence protocol by 12% (average) and reduces L1 cache miss rate by 17%. It also reduces the network power consumption by as much as 26% (5% in average) and the network latency as much as 10%.

Keywords: MOESI; Cache Coherence Protocol; last touch time; reuse frequency; core locality;

1 Introduction

The current trend in chip multiprocessor (CMP) is to incorporate large number of cores. The earlier implementation of multi-core technology integrated small number of cores (two to four) in to the CMP. But now the industry is heading towards larger number of cores on each processor to enhance the throughput. For example, IBM Cell Processor has eight cores [14], and Tilera TILE64 has [15], 64 cores incorporated in a single chip. Along with these technical trends, the cache coherence complexity has also increased significantly.

Cache coherence is the mechanism that allows maintaining the state and value of a cache block in the caches of a chip multiprocessor (CMP). It ensures that no data is lost or overwritten before the data is transferred from a cache to the target memory. Since cache coherence involves significant amount of communication over wires, wire speed and bandwidth are the major concerns for the improvement of the performance and scalability of the cache coherence protocol. Due to the fact that, in a CMP the interconnection speed does not scale well enough with the increase in core speed, smart mechanisms and novel policies are necessary for accelerating the performance of the coherence protocols. Chip multiprocessors with a small number of cores can use a snoop-based coherence protocol which relies on a broadcast-based interconnect [1]. However, broadcast based interconnect uses the bus which degrades the performance as the number of cores increases. As snoop-based protocol lacks scalability, and therefore directory based protocol [2] is widely used in a large scale CMP. The basic idea is to keep a directory entry for every memory line. This entry consists of its state and a sharing code indicating the caches that contain a copy of the line, which are termed as sharers of a cache block.

On a cache miss, each coherence transaction is sent to a directory controller which, in turn, using its corresponding entry, redirects it to the cores caching the line, e.g., redirects to the sharers of the cache line.

This paper examines the design of effective coherence mechanisms for a directory-based multi-core architecture that has a shared last level cache (L2 cache as LLC) which maintains MOESI (Modified, Owner, Exclusive, Shared, Invalid) [5] cache coherence protocol and ensures the correctness of the cache coherence substrate via token counting. In this work, the directory-based MOESI token protocol is used instead of the broadcast-based token protocol (TokenB) [1] to enhance the scalability of the protocol. We begin by exploring the directory-based MOESI token cache coherence protocol (MOESI_CMP_Token) for multi-core processors in a CMP and adapt certain mechanisms to attune the directory based MOESI protocol for boosting the performance in terms of execution time, power consumption, and network latency.

The conventional MOESI coherence protocol is a write-invalidation based protocol and has no mechanism for updating the sharers. In a pure write-update cache coherence protocol, all the sharers are updated according to the new
modified value whereas in a write-invalidation based coherence protocol the existing sharers of the cache block are not updated according to the new value. Previous researches [13] [22] have already shown that pure write-update is highly undesirable because of the heavy traffic caused by the updates. It has also been shown that in a pure write-update protocol the network traffic increases with the increase of the CMP size [22]. In this paper we show that simply implementing write-invalidate based directory protocol does not provide the most effective solution. In our proposed mechanism, we keep record of the access frequencies of the cache blocks, e.g., the number of times a cache block is reused in the past, and term this metric as reuse frequency (RF). We also consider the time-based aging of the accesses of the cache blocks for better speculation of reuse in the near-immediate future and have termed the last access time as last touch time. In our work, we have incorporated the concept of temporal locality of a cache block by means of its reuse frequency and last touch time. Many of the cache blocks in the shared LLC may show very high temporal locality while the others may not. If we can update the sharers of the cache blocks which show high temporal locality, then many of the requests of the local cores can be satisfied by the updated sharers’ content. This will eventually improve the performance of the cache coherence protocol in terms of execution time and power consumption.

To the best of our knowledge, our proposed work, Exploiting Reuse-Frequency with Speculative and Dynamic Updates in an Enhanced Directory Based Coherence Protocol (RFU-Dir), is the first one that applies a smart mechanism policy to dynamically update the sharers of only those cache blocks who are highly reused in the recent past. The updates are dynamic because after each and every write back of an individual cache block in the LLC, the decision to update is taken dynamically by the Dynamic Speculative Update Mechanism Policy (DSP). In this work, we show that RFU-Dir enhances the speed up of the directory based MOESI cache coherence protocol by 12%, decreases L1 miss rate by 17% (average) and reduces the L1 miss latency by 3.2% (average). Our proposed work also decreases network latency and L1 replacement as much as 10% and 90% respectively. The network power consumption and the directory access are also decreased by 5% (average) and 13% (average) respectively.

The remainder of the paper is structured as follows. Section 2 identifies the motivation of our work, Section 3 presents the dynamic speculative update mechanism and Section 4 presents the methodology with the experimental results. Related work is summarized in Section 5. Finally, Section 6 outlines the main conclusions of this work.

2 Motivation

Previous researches have shown that, if a cache block is used by a core in a CMP, then there is a high possibility that the particular cache block will be reused by that core in the near-immediate future. In other words, if at time $t_0$, if core1 issues an address for a cache block ‘A’ then within a near-immediate future time interval, $t_0 \leq t \leq t_0 + k$, where $k$ is an integer, there is a high possibility that core1 will again issue request for that cache block ‘A’. In our work, we term this phenomenon as core locality for a cache block residing in the LLC. The conventional pure update-based protocol updates the sharers of each and every cache blocks [13]. Updating the sharers of each and every cache blocks demands excessive bandwidth which has already led the update-based protocol to virtual extinction, especially in a broadcast-based snoopy protocol [13]. This situation gets worse as large number of cores is incorporated in a CMP. On the other hand, a pure write-invalidate protocol does not take advantage of the updates of the potential sharers which may show very high temporal locality. In order to overcome these limitations, we make use of the reuse frequency and last touch time of each cache block as criteria for the sharers to be updated or not.

To understand the motivation of this work, let us now assume a scenario where a READ (L1_GETS) request is issued for a cache block ‘A’ by core1in a 64-core CMP. Due to a read miss in the local core, the request (L1_GETS) is sent to the shared LLC and the desired data is sent back to the local cache of core1. Let us assume that within a short interval of time, in the same way core3, core5, and core19 also have issued L1_GETS request for the same cache block ‘A’ and eventually all these cores have cached a local copy of the block ‘A’. So core1, core3, core5 and core 19 are now the sharers of the cache block ‘A’ residing in the LLC.

Now, let us assume another scenario that, at time $t = t_0$, core17 has issued a WRITE request (GETX) for ‘A’ in the LLC and at time $t = t_0 + n$, where $n$ is an integer, the modified cache block is written back to the LLC. In the write-invalidate protocols, none of the sharers are updated after the write back to the LLC. However, due to the existing tendency of core locality, there is a high possibility of a core to reissue request for the same cache block in the near-immediate future. So in the near-immediate future, if core 1, core 3, core 5 and core 19 again reissues request for cache block ‘A’ then all these requests will result in coherence misses. This will trigger many message passing transactions. This will increase the power consumption, network latency, bandwidth contention and the execution time. The situation may get worse as the size of the CMP gets larger. A novel and smart update mechanism policy can improve the performance of the CMP by considering the limiting factors of the pure update protocol. In our work, we have updated the sharers of only those cache blocks which have more potential to be reused in the near future and ignore the updates of those sharers which are less probably to be used in the future.

3 Dynamic speculative update mechanism

3.1 Speculation of high reuse of cache blocks via reuse frequency and last touch time

High reuse frequency and last touch time in the near-immediate past enhance the possibility of a cache block to be reused in the near-immediate future. So, in our work we have considered these two metrics to dynamically update the
sharers. To increase scalability we have considered the directory based coherence protocol instead of the snoopy based protocol, to minimize the bandwidth overhead with the increase in the number of cores. When a cache block is not present in their local caches then the request (L1_GETS) is forwarded to the shared LLC. This situation is depicted in Figure 1. In our work, each cache block in the LLC is associated with a reuse frequency counter which is initialized to zero and is reset when the cache block is evicted from the LLC. The reuse frequency counter is incremented with each request that comes to the LLC for that cache block. A reuse frequency counter achieves high value when several cores have issued requests for the same cache block. The high reuse frequency of a cache block in the shared LLC also depicts the fact that a particular LLC cache block has high possibility of having multiple sharers. Again, due to core locality, there is a high possibility that the same core reissues requests for a cache block within a short interval of time. For this reason, if the sharers are updated based on the last touch time then the performance can also be improved due to the tendency of temporal locality of the cache blocks.

![Figure 1. Several READ Requests (L1_GETS) is coming from different L1 caches to the same cache block in the LLC because cache miss occurs in their corresponding L1 caches. Thus increasing the reuse frequency of the cache block in the LLC.](image)

3.2 Implementation of dynamic speculative update mechanism policy

In this work, we introduce an intelligent and novel scheme of speculative update of the sharers which enhances performance of the write-invalidation based directory protocol in terms of execution time, network latency and power consumption. In our proposed work, wherever a write back occurs in the LLC, the sharers are not updated for every cache block. Instead the Dynamic Speculative Update Mechanism Policy (DSP) is being consulted. The sharers are updated only when the cache block is being reused again and again in the past and has a last touch time in the near immediate past. Figure 2(a) depicts the scenario of consulting the DSP and 2(b) depicts that after the condition checking of the reuse frequency and the last touch time, all the sharers are being updated. The constraints of the Speculative Dynamic Update Mechanism Policy are described below:

Speculation of reuse via high reuse frequency: The reuse frequency of the cache block, whose sharers are to be updated, should be greater than the Threshold value (experimental value of Threshold=13) to assure the higher possibility of being reused in the future, i.e. Reuse Frequency > Threshold.

Speculation of reuse via last touch time: A cache block may be reused again and again long before in the distant past, but may not be reused in the near immediate future. Updating the sharers of such a cache block with high reuse frequency may cause a hazard to performance of the CMP. Our work ensures the reusability of a cache block in the near immediate future by considering its last touch time. A cache block which is touched in the near immediate past has a high possibility of reuse in the near immediate future due to its temporal locality. In this work, the life span time of a cache block is divided into many parts, such as, near immediate past, near immediate future, distant past. We define the near immediate past as, \( t \leq t_0 \leq t \leq t_0 + k \), where \( k \) is a integer (experimental value of near immediate past is time range of 2000 microsecond in the past from the current touch time of a cache block) and distant past as, \( t < t_0 \) (experimental value of distant past is more than 2000 microsecond time span in the past from the current touch time of a cache block).

3.3 Space conflict for the speculative update

In our proposed work, when a L1 cache receives a speculative update and it does not have enough space to hold the line then the LRU line of that L1 cache is evicted. The LRU line is evicted instead of cancellation of the update request because the cache blocks with high reuse frequency has a higher possibility of reuse in the near future compared to the least recently used block.

3.4 Extra hardware

Reuse frequency counter is used to keep record of the reuse frequency. Each cache line is associated with a reuse counter and a 32 bit buffer is used to keep track of the last touch time. The reuse frequency counter is assigned an initial value of zero and is incremented when a request to that cache block is received. For calculating the reuse frequency, we used a 4 bit counter per cache line. Therefore, extra hardware cost for counters:

\[= \frac{\text{(#of cache lines} \times \text{counter size})}{\text{LLC size}}\]

\[= \frac{[(1 \text{ bank size}) \times \text{(countersize)/ (line size)}/ \text{(8 bits/counter size)}]}{[256 \times 1024 \times 8 \times \text{# of banks}]}
\]

\[= \frac{[(1256 \times 1024) \times 64 \times 8 \times 4]}{64 \times 8} / [256 \times 1024 \times 8 \times 64] = 0.007 \text{ (0.07% of total LLC size).}\]

4 Evaluation methodology and results

In this section, we first describe our experimental setup. Then we present the experimental results for our proposed model validation. We contrast the proposed work with the existing invalidation based MOESI token coherence protocol (MOESI_CMP_Token) to verify the impact of our proposed work in terms of performance evaluation.
4.1 Experimental setup

In order to evaluate the proposed RFU-Dir scheme, we set up a NUCA-based L2 cache model and the parameters are listed in TABLE I. We evaluate our system using the Virtutech Simics full system execution-driven simulator along with GEMS [4] timing model. The network latency and power is measured by GARNET simulator [6] and Orion [7] simulator respectively. To simulate real applications’ L1 and L2 cache behavior, we ran PARSEC 2.1 benchmark suit.

![Image](image_url)

2(a). The Cache Controller of the L2 bank in the LLC is consulting the Speculative Update Mechanism Policy whether the sharers should be updated or not.

2(b). Mechanism of updating the sharers in RFU-Dir.

Figure 2. Sharers Update Mechanism in RFU-Dir.

We assume a MOESI token based cache coherence protocol (MOESI_CMP_Token) among the L1 caches and L2 caches. We also assume a two level cache hierarchy where L2 is the shared Last Level Cache (LLC) shared by 64 cores in a Chip Multiprocessor (CMP). All the L1 caches are local to the cores. So there are 64 L1 caches. L1 caches are split into L1-I (instruction) and L1-D (data) caches. In the protocol, the L2 cache is a Non Uniform Cache Architecture (NUCA) based model [17] and has 64 banks. We assume the protocol to be directory based which has 64 directories. Since our primary focus is to reduce the coherence miss rate, improve the execution time and reduce the power consumption, this processor model is sufficient to evaluate our scheme’s fundamental performance. We consider the applications from the PARSEC 2.1[8] benchmark suites. We used the default input set and sim-large configuration for PARSEC 2.1 applications. PARSEC 2.1 applications were run to completion starting from the beginning of their parallel sections and the applications were run for a billion instructions starting from their region of interest (ROI).

4.2 Experimental results

In this section, the results of the comparative evaluation between the conventional directory based MOESI coherence protocol and our proposed work is presented. We considered the benchmarks from the PARSEC benchmark suit that has a certain number of sharers [8]. We did not consider the benchmarks that have no sharing patterns. In our proposed scheme, the threshold value of reuse frequency (RF) and the near-immediate past are varied along a range of values but in our experiment, we observe that reuse frequency=13 gives the most optimal result in most of the cases. The value of the most effective reuse frequency depends upon the benchmarks that are used. In our case, when the reuse frequency is a very small value, e.g. reuse frequency =3, then the result is not enhanced. This is because of the fact that when a cache block is touched only a fewer number of times, e.g. 3 times, then the possibility of reuse of the cache block is much lesser in the future. So, if we update sharers of the cache block with low reuse frequency then the performance will not be improved. Again, when the reuse frequency increases then the possibility also increases that the cache block will be reused in the future. In our experiment, when reuse frequency is selected within a range 10 to 13 the performance is improved. However, increasing the reuse frequency value greater than 13 degrades the performance because in this case the cache blocks are overly utilized and has a very poor possibility of reuse in the future. So updating the sharers increases the execution time and the network power consumption. The near-immediate past is considered as an approximate time range of 2000 microsecond in the past.

**TABLE I. PARAMETERS FOR THE SIMULATED PROCESSORS**

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Chips</td>
<td>1</td>
</tr>
<tr>
<td>Number of Cores per chip</td>
<td>64</td>
</tr>
<tr>
<td>Processor Model</td>
<td>In-order</td>
</tr>
<tr>
<td>OS</td>
<td>Solaris 10</td>
</tr>
<tr>
<td>CPU</td>
<td>Ultraspar-iii-plus</td>
</tr>
<tr>
<td>Main Memory Latency</td>
<td>200 cycle</td>
</tr>
<tr>
<td>Latency</td>
<td>1 cycle</td>
</tr>
<tr>
<td>L1 Cache Set Associativity</td>
<td>4</td>
</tr>
<tr>
<td>L1 Cache Block Size</td>
<td>64B</td>
</tr>
<tr>
<td>L2 Cache Set Associativity</td>
<td>4</td>
</tr>
<tr>
<td>L2 Cache Block Size</td>
<td>64B</td>
</tr>
</tbody>
</table>

4.2.1 Analyzing cache miss rate and its consequences

Our proposed work decreases the L1 cache miss rate by 17 % (average) by dynamically updating the local copies which have higher possibilities to be reused in the near
immediate future. So the possibility is high that when the requests are issued for those data blocks they could be satisfied by their local caches. Thus it decreases the L1 cache miss rate. Figure 3 shows the comparison of the L1 cache miss rates (MPKI -Misses per Kilo Instructions) of the conventional MOESI protocol and our proposed work, RFU-Dir. From Figure 3, we can see that for the benchmarks bodytrack and fluidanimate, the L1 cache miss rate decrease by 30% and 35% respectively. Ferret, freqmine, raytrace, swaptions and vips show a cache miss rate reduction of 8%, 4%, 5.5%, 2% and 2.5% respectively. In this work, we have updated the sharers of the highly utilized cache blocks so that they may be reused. Here, we can observe that the reduction of the L1 cachemiss rate is larger for fluidanimate than for vips since fluidanimate has larger degree of sharing patterns than that of vips. The number of sharers of the cache blocks varies from one application to another. Some applications have a higher degree of sharing pattern while others have a lower degree. In our work, those applications whose cache blocks have larger number of sharers contribute more cache hit rate. When a data block is requested by the core and it is not present in the local cache then the request is sent to the shared LLC. After finding the data from the lower level cache (LLC) or from the main memory the data block is sent to the local cache. If the local cache does not have enough space to hold the data the Least Recently Used (LRU) block is then replaced by the incoming data block. It can be observed that our work results in increased L1 cache hit rate. This means that many of the issued requests are satisfied by the L1 caches. This minimizes the cache block requests to the LLC. As a result, the total numbers of L1 replacements are decreased. Figure 4 shows the percentage of the change in L1 replacements in our proposed RFU-Dir. Our proposed work decreases the replacement of the cache lines of the L1 cache by 44% in bodytrack, by 4% in fluidanimate, by 3% in freqmine, by 90%, raytrace 29%, and by 6% in swaptions. However, vips does not show any improvement as the cache blocks of vips exhibits less temporal locality.

We also notice that the READ (L1_GETS) requests are issued to the LLC as a result of the cache misses in the local caches. In our proposed work, many of the requests issued by the cores are satisfied by the updated values of the sharers residing in the L1 cache. The decrease in the number of the READ (L1_GETS) requests, coming from the L1 caches to the shared LLC, is one of the consequences of the increase in the hit rate of the L1 caches. Figure 5 shows the decrease of the total number of READ (L1_GETS) requests coming from L1 caches to the shared LLC.

4.2.2 Analyzing execution time

Speculative and dynamic updates of the sharers increases the L1 cache hit rate which eventually decreases the directory accesses and network latency. Directory accesses occur on the consequences of cache misses which cause long latencies and extra performance overhead. The increased L1 cache hit rate is able to minimize many unnecessary transactions such as ACK, SEARCH_DATA, SEND_DATA, and NACK which also minimizes the average network latency. DSP also decreases the L1 miss latency. The decreases of these parameters enhance the speedup of our work. Figure 6 shows the increase in speedup of each benchmark. From the graph, we can observe that our proposed work maximizes the speedup of fluidanimate and bodytrack. We have observed that the LLC cache blocks of bodytrack have a large number of sharers, than many other benchmarks [8]. So, constraints of the DSP cause a significant improvement in the speedup of the benchmark. Ferret, freqmine, raytrace, swaptions and vips show a certain degree of cache blocks’ sharing pattern [8]. For this reason, our work increases the speed up of these benchmarks to a certain degree in compare to fluidanimate. The execution time is also proportional to the total latency of the network. If the network latency of a CMP can be minimized then it also influences the total execution time. Figure 7 shows the comparison of the average network latency.

![Figure 3. Comparison of the decrease in L1 cache miss rate (MPKI).](image)

![Figure 4. Percentage of change in L1 replacements in RFU-Dir in compare to the Convensional directory based MOESI Cache Coherence.](image)

![Figure 5. Comparison of the number of READ requests (normalized to total number of instructions) coming to the shared LLC as results of L1 misses.](image)

![Figure 8. Comparison of the normalized number (normalized to the total number of the instructions) of directory access for tag searching.](image)
the percentage of decrease in L1 miss latency in RFU-Dir. All these factors contribute to the improvement in the speed up of RFU-Dir.

![Increase of Speedup in RFU-Dir](image1.png)

Figure 6. Increase in Speed up of RFU-Dir in compare to the conventional Directory based cache coherence (MOESI_CMP_Token).

![Average network latency](image2.png)

Figure 7. Comparison of the average network latency (microsecond).

![Normalized Number of Directory Accesses](image3.png)

Figure 8. Comparison of the Normalized number of Directory Accesses (normalized to total number of instructions) in Directory based cache coherence (MOESI_CMP_Token) and RFU-Dir.

### 4.2.3 Analyzing power consumption

Updating the sharers speculatively and dynamically improves the hit rate which in turns decreases the total number of message passing mechanisms which would require extra clock cycles. All these activities may include the searching of the tag array in the L1 cache, LLC and also in directory, the initiation of the requests of the transactions, sending DATA to the requestor, acknowledgements (NACK, ACK) etc. All these activities not only increase the network latencies and occupy the precious on-chip bandwidth but also increase the network power consumption. Figure 10 shows the change of the network power consumption in RFU-Dir compared to the conventional MOESI_CMP_Token. It can be observed that as the number of sharers in vips is less the locality of the reused cache blocks is not dominant.

![Percentage of Change L1 Miss Latency](image4.png)

Figure 9. Percentage decrease of L1 cache miss latency in RFU-Dir in compare to the conventional Directory based cache coherence (MOESI_CMP_Token).

### 5 Related works

Previous works in cache coherence have included prediction, which have covered a broad spectrum. In [18], Eggers and Katz compare write-invalidate and write-update snoopy-cache protocols. Eggers and Katz also evaluate two extensions to pure write-invalidate and write-update called protocol read-broadcast and competitive snooping [19]. Eggers and Katz’s competitive snooping protocol is an implementation of Karlín’s Snoopy-Reading protocol [20]. Grahn et al. [22] proposes competitive-update and showed that pure write-update is highly undesirable in the general case because of the heavy traffic caused by the updates. Their work focus on setting a competitive-threshold value to the counter associated to each cache block in the local cache and the counter is decremented upon getting an update message from a remote processor. When the counter reaches to zero the copy of the cache block is invalidated and further updates to that cache block are ceased. There are significant differences between Grahn et al.’s work and our proposed work. In our work we assign the reuse frequency of each cache block in the shared LLC as zero and increment the value upon each access to that cache block. In our work we have considered the reuse frequency as well as the last touch time of the cache blocks to take the decision dynamically whether the sharers should be updated or not. In [22], Archibald proposed an adaptive write-invalidate/write-update snoopy-cache protocol. In their work they invalidated all other copies of the block when a single processor has issued three consecutive writes to the same block without any intervening access by another processor. Some of these techniques [9] [11] also added states to the coherence protocol. Some of the protocols focused on table based predictors [3] [10] that predicted messages before their arrival and took necessary steps at proper time.

In the previous works several coherence protocols [9][12] are proposed but an intelligent and novel scheme for a directory based protocol has not been studied where the reuse frequency and last touch time of a cache block are used to speculatively update the sharers. The dynamic and speculative updates of the sharers have improved the performance of RFU-Dir. We implement RFU-Dir on the directory based protocol to enhance scalability and ensure low traffic.
overhead. We use simple counters and buffers for keeping track of the reuse frequency and last touch time for each cache block respectively.

Figure 10. Decrease of power consumption in RFU-Dir (in percentage).

6 Conclusions

We proposed a speculative-based dynamic update mechanism to update the sharers dynamically in the conventional directory based MOESI coherence protocol. The proposed dynamic update mechanism is not limited to MOESI cache coherence protocol. It can also be applied to cache coherence protocol like MOSI (Modified, Owner, Shared, Invalid), and MSI (Modified, Shared, Invalid). Our dynamic and speculative update mechanism can be applied in any invalidation directory-based protocol. By using low overhead counters and buffers we keep track of the reuse frequency and the last touch time of the cache blocks respectively and instead of updating the sharers of all the cache blocks in the LLC we update the sharers of only those cache blocks which are highly reused in the near-immediate past.

We showed that simple counters and buffers can effectively identify a substantial portion of the sharers which may show temporal locality in the near-immediate future. Thus, updating those sharers result in improvement in the execution time, power consumption and hit rate. Our work also reduced the network latency, the L1 miss latency, the total number of L1 cache replacements and the directory accesses for searching the tag arrays. In the future we aim to further improve the coherence protocol in terms of performance, power, protocol complexity and scalability.

7 References


