Parallel Computer Architecture

ECE 382

Coherence Prediction in Parallel Computers

Author: Preston Glenn, Jason Ho, Madison Threadgill

November 3rd

Motivation

In the quest for better performance, prediction mechanisms have become a widely studied field in computer architecture. The idea is simple. If we can correctly predict the future, we can use that information to unblock dependencies which enables greater parallelism. Such examples include branch predictors which try and predict the direction of branches without finishing other dependent instructions in the pipeline, cache replacement policies which implicitly predict which cache lines are least likely to be used in the cache, and prefetchers which predict which cache lines the processor would need in the future. These mechanisms all work in the microcosm of a single core which works when programs share no data, but when it comes to multiple processors that need to share memory, there are additional prediction mechanisms that can be explored.

In a multi-processor distributed shared memory (DSM) system, main memory is physically split between all nodes even though the processor's view of the shared memory is one contiguous memory. This computing paradigm has become very common in servers which combine many computers together. On each node's main memory, a node's directory must maintain coherence between all processes by keeping track of which processor has access and what the state of the memory is at all times. To get access to data that is not local to a processor, it must incur remote access latency on the distributed network to fetch the correct data and update old data. Similarly, if a directory needs to invalidate data to provide modify permissions to a processor, it requires all sharers to acknowledge the invalidate and clean their data, incurring remote access latency in both directions as well as the amount of time it takes to process the request on both sides. After we have gotten acknowledgments from all sharers, the home site can fulfill the original request. We illustrate the worst case scenario in figure 1, in which the original requester must incur at least four separate remote access latency requests to get permission to modify a previously shared line.

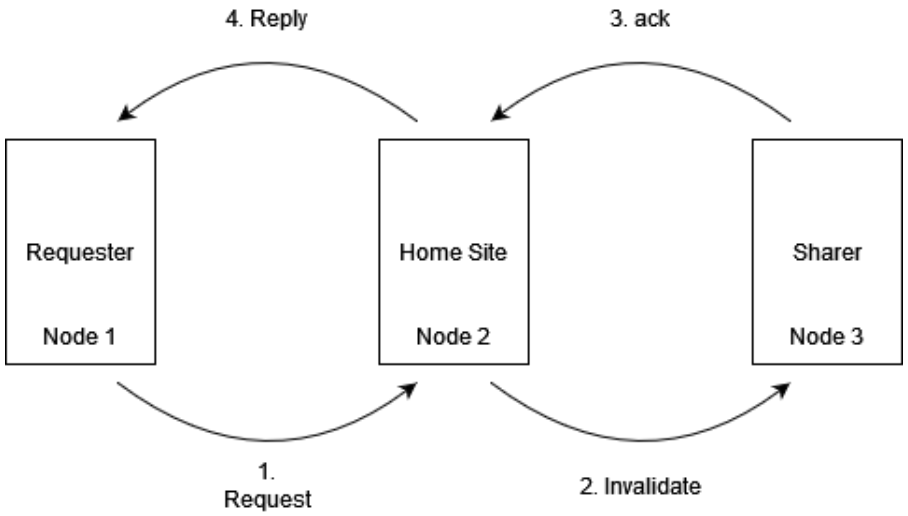

Figure 1: Worst Case Number of Messages in DSM to maintain coherence

To improve DSM performance, it is natural to consider whether we can hide these remote access latencies. One such approach comes from Mukherjee and Hill in 1999, who introduced the idea of predicting the next coherence message using a two level predictor from the running history of coherence messages for a cache line on both the directory side and the cache side [4]. If all coherence messages are predicted correctly, the worst case becomes the average case, as the home site can immediately service a request as all other sharers have already invalidated themselves by predicting that this write would come. Predicting all next coherence messages is dangerous though, as a processor may predict that it will get modified access before it comes, which may leave the directory in an invalid state. Similarly, Lai and Falsafi found it unnecessary and wasteful to try and predict acknowledgment coherence messages [2]. To this end, it is not the coherence messages that should be predicted, but rather, the future coherence state of cache lines on the processor side and the directory side. On the processor side, it needs to predict whether it is done using a cache line based on previous sharing patterns, such that it can invalidate itself early. This could be described as a coherence-aware cache replacement policy. On the directory side, it should predict which processors needs access to a line after a write invalidation. This could be described as a coherence-aware prefetcher. This producer-consumer problem was most recently examined in 2006 using a perceptron-based solution, another branch predictor by Leventhal and Franklin [3].

Now, more than twenty years since the original problem was introduced by Mukherjee and Hill, shared memory multiprocessor servers and systems have become even more prevalent in the computing domain, yet there has been little to no work in this area since 2006. In that time, branch predictors, cache replacement policies, and prefetchers have become much more advanced. This begs the question of whether it is possible to apply the knowledge of these new prediction mechanism systems to produce better coherence predictors for shared memory workloads, just like how the two level branch predictor and perceptron branch predictor were originally adapted for this problem. At the same time, shared memory workloads have changed, so it also makes sense to analyze the impact that coherence message predictors can have on modern benchmarks.

Related Works

The original paper on this problem comes from Mukherjee and Hill in 1998, who proposed Cosmos, an adaptation of Yeh and Patt's two level branch predictor [6] for general coherence message prediction at both the directory side and the cache side [4]. Previous solutions to predict sharing patterns required prior knowledge before runtime to augment the coherency protocol, but this solution could be trained as the program progressed and sat as an addition to the coherency protocol. With Cosmos running on five scientific applications of the time, it achieved next-message coherency message prediction accuracy from 63 % to 93 %, suggesting that coherency message prediction is possible without requiring much more area than a two level branch predictor at the directory and at the processor cache.

The second paper comes from Lai and Falsafi in 1999 and build directly off of the work of Cosmos from the year prior [2]. They propose that not only is acknowledgment prediction

unnecessary, but that it can also be detrimental. This is due to the fact coherence messages can arrive in arbitrary order. Thus if one was to remove the acknowledgements from the pattern tables one would be able to "substantially reduce perturbation in the tables due to message re-ordering" and thus improve performance. Additionally, the paper proposes a Vector Memory Sharing Predictor (VMSP). Here, they propose using a vector map to encode sequences of read requests to identify multiple readers of a block while disregarding read order, which allows them to greatly reduce the number of entries in the pattern table. With these two improvements, the authors saw improvement in accuracy from 81 to 86% to 93% respectively on a variety of SPLASH-2 benchmarks.

In the near future, also in 1999, Kaxiras and Young published a third paper on the topic, which provided a taxonomy of previous prediction schemes related to coherence and how to categorize a good coherence predictor given the drawbacks of misprediction for false positive and true negatives [1]. With this methodology, they propose that the largest benefit of coherence prediction comes from consumer prediction, concluding that determination of potential readers is best predicted after a write invalidation. Moreover, the results suggest that using a hybrid approach of indexing using both a memory address and/or the program counter can increase performance. This approach combines instruction locality with address locality to predict the coherence state, which implies that coherence traffic is often unique to both the instruction and memory address. Finally, this work finds that there is an essential trade-off between latency and bandwidth usage in coherence prediction protocols. Sensitive coherence prediction schemes generate extra traffic in the network, increasing the latency of the network if bandwidth-bound. Therefore, researching more accurate coherence prediction mechanisms that use bandwidth conservatively is essential.

The last work comes from Leventhal and Franklin in 2006, who proposed an augmented percepton branch predictor targeted at determining consumers of data after a write invalidation [3]. Similar to branch prediction, the use of a perceptron scheme allows them to use more history to make predictions as the size of the history scales linearly, while in a scheme such as two level branch prediction, increasing the length of history scales exponentially. They show that while a processor's own history is the strongest indication of whether it will be a sharer the next time a write invalidation happens, there is also some correlation with other processor histories as to whether this line will be used by a processor. They use perceptrons to exploit the relative weighting of other processor history to determine whether that processor will be a sharer.

While the main contributions to coherence protocol predictions have halted, there are a wide variety of new approaches to consider from other prediction mechanism techniques that coherence prediction methods first derived from. For example, coherency prediction can leverage state-of-the-art branch prediction methods or prefetcher history storage mechanisms to make smaller, more accurate predictors that hide remote access latency to directories.

Proposed Approach

Our proposed approach to tackle this research question comes in three steps.

1. **Obtaining Coherence Traffic Traces of Benchmarks** The previous work uses SPLASH and SPLASH-2 benchmarks, which are approximately twenty years old [4, 2, 1]. These benchmarks may not reflect today's common parallel workloads such as machine learning, cloud computing, scientific workloads, and server operations. Obtaining such workloads would be pivotal to first determine whether there is headroom to innovate. For instance, Kaxiras and Young found that in the SPLASH benchmarks, they may not have been large enough to sufficiently stress the predictive sharing schemes as only approximately 15% of cache lines used in the program were shared [1]. It is not clear if new benchmarks and workloads such as SPLASH-3, PARSEC, or even basic database operations like SQL server traces would show more shared cache line contention. Starting with PARSEC, we will use Intel's PIN tool to extract coherence messages and data from a multi-core system. Time permitting, we will also examine other benchmarks such as SPLASH-3, and even write database synthetic workloads.

2. **Characterizing Coherence Traffic Traces** Once we have traces from various workloads, we will first need to characterize them. That is, we will determine how much shared memory traffic takes place per benchmark. In particular, we will look into the total percentage of cache lines that are shared, how many sharers are on a line after a write invalidate, and other metrics to measure the headroom possible by predicting coherence traffic on modern benchmarks. To do this, we will need to parse through the traces and count the various occurrences with Python or some other scripting language.

3. **Leveraging Prediction Mechanisms and Applying to Coherence Prediction** We plan to focus on a two pronged approach for coherence prediction. That is, consumer prediction after write invalidates on the directory side, and invalidation prediction on the processor / cache side. While previous work has leveraged coherence predictors at both locations as well, they are typically assumed to have the same data, but this is not necessarily true. On the processor side, for instance, keeping track of previous memory access patterns (that then result in an invalidate on a line) is much easier to do. Similarly, on the directory side, it does not make sense to have access to instructions if the directory is remote. We plan to start by leveraging research in more modern prefetchers and branch prediction and determining if these methods can be used for cache coherency prediction. For example, the work in [5] claims that prefetching data (as well as coherence data) can get stale quickly, causing accuracy faults, especially for lines that are not used for a lengthy period of time. Naturally, this extends as a orthogonal way of organizing history from two level branch prediction by using a index table and a global history buffer (GHB). Indexing could be performed by a combination of memory address and instruction address (depending on the location of a coherence predictor), obtaining a coherence message stream for that address from the index. This can then be used to predict the next coherence state. The GHB is just one example of a theoretical better way to organize data while getting rid of stale data early. Similarly, there has been continued work on perceptron branch prediction, as well as the use of

perceptrons in other prediction mechanism schemes. These ideas in different domains will serve as starting points and iteration for coherence prediction.

Completely orthogonal to previous hardware-based prediction mechanism solutions, time-permitting, we plan to determine an upper-bound limit for accuracy using completely hardware unconstrained methods, such as deep neural networks. These solutions may even guide our exploration of hardware-constrained solutions by revealing the relative importance of different inputs by evaluating the weights of inputs.

References

[1] S. Kaxiras and C. Young. "Coherence communication prediction in shared-memory multiprocessors". In: *Proceedings Sixth International Symposium on High-Performance Computer Architecture. HPCA-6 (Cat. No.PR00550)*. HPCA: 6th International Symposium on High-Performance Computer Architecutre. Touluse, France: IEEE Comput. Soc, 1999, pp. 156–167. ISBN: 978-0-7695-0550-3. DOI: 10.1109/HPCA.2000.824347. URL: http://ieeexplore.ieee.org/document/824347/ (visited on 10/30/2023).

[2] An-Chow Lai and B. Falsafi. "Memory sharing predictor: the key to a speculative coherent DSM". In: *Proceedings of the 26th International Symposium on Computer Architecture (Cat. No.99CB36367)*. Proceedings of the 26th International Symposium on Computer Architecture (Cat. No.99CB36367). ISSN: 1063-6897. May 1999, pp. 172–183. DOI: 10.1109/ISCA.1999.765949. URL: https://ieeexplore.ieee.org/document/765949 (visited on 11/01/2023).

[3] Sean Leventhal and Manoj Franklin. "Perceptron Based Consumer Prediction in Shared-Memory Multiprocessors". In: *2006 International Conference on Computer Design*. 2006 International Conference on Computer Design. ISSN: 1063-6404. Oct. 2006, pp. 148–154. DOI: 10.1109/ICCD.2006.4380808. URL: https://ieeexplore.ieee.org/abstract/document/4380808 (visited on 11/02/2023).

[4] Shubhendu S. Mukherjee and Mark D. Hill. "Using Prediction to Accelerate Coherence Protocols". In: *Proceedings of the 25th Annual International Symposium on Computer Architecture*. ISCA '98. Barcelona, Spain: IEEE Computer Society, 1998, pp. 179–190. ISBN: 0818684917. DOI: 10.1145/279358.279386. URL: https://doi.org/10.1145/279358.279386.

[5] K.J. Nesbit and J.E. Smith. "Data Cache Prefetching Using a Global History Buffer". en. In: *10th International Symposium on High Performance Computer Architecture (HPCA'04)*. Madrid, Spain: IEEE, 2004, pp. 96–96. ISBN: 978-0-7695-2053-7. DOI: 10.1109/HPCA.2004.10030. URL: http://ieeexplore.ieee.org/document/1410068/ (visited on 02/13/2023).

[6] Tse-Yu Yeh and Yale N. Patt. "Two-Level Adaptive Training Branch Prediction". In: *Proceedings of the 24th Annual International Symposium on Microarchitecture*. MICRO 24. Albuquerque, New Mexico, Puerto Rico: Association for Computing Machinery, 1991, pp. 51–61. ISBN: 0897914600. DOI: 10.1145/123465.123475. URL: https://doi.org/10.1145/123465.123475.