Coherence Protocol Optimization for Private and Shared Data

Nipun Katyal
nkatyal@andrew.cmu.edu

Chen Fu Hsu
chenfuh@andrew.cmu.edu

May 2024

1 Summary

In order to counter the limitations of Moore's law where hardware performance can't be improved vertically (faster hardware) due to physical constraints, an orthogonal approach to increase the number of processors(threads) has proven effective. However, horizontal scaling (more processors) comes with own problems, simply increasing the number of processors won't help if memory becomes the bottleneck. Adding a dedicated cache for each processor will ease the pressure on the shared caches but requires additional work to keep all the caches in sync with the global state. Coherence protocols come in two flavors - Snooping-based and directory-based. Snooping-based protocols require minimal changes to the cache structure to keep track of the cache lines but incur an expensive broadcast request to all the other caches to invalidate/request data. Hence, such protocols don't scale well with the increasing number of physical threads. Directory-based protocols omit the problem by maintaining some additional state about the sharers of data which can be looked up during invalidation/request. This transforms the broadcast to point-to-point lookup but uses costly cache space. This project aims to find the middle ground for Snooping- and directory-based approaches by keeping the same cache structure as a snooping-based protocol would but introducing a bloom filter hardware component to keep track of sharers, which can then be used to perform a point-to-point lookup as in a directory based protocol.

2 Background

Given the vastness of processor design, we would like to start with the description of the hardware configuration for which we propose the coherence protocol optimization. The diagram below shows a Chip Multi-Processor with 4 cores. Each core has a dedicated L1 cache and all the cores share a common L2 cache connected through a ring or bus interconnect.

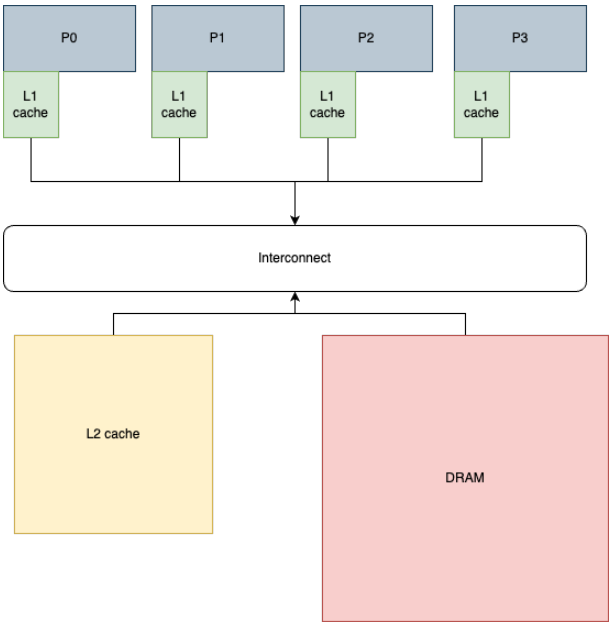

Figure 1: Component organization

2.1 Snooping-based protocols

Snooping-based protocols rely on a shared bus that connects all the caches and the DRAM to form a coherent system. All the requests are observable to the components on the bus and each cache or DRAM responds according to the state of the cache line pointing to requesting address. The list below is a non-exhaustive collection of snooping-based protocols that were experimented with in our cache simulator.

- MESI protocol associates each cache line with 4 states Modified(M), Exclusive(E), Shared(S) and Invalid(I). A cache line in the modified state is the only valid line for that address in the same level. Any read/write requests for the address by the same processor are responded from the local cache while such requests from the other processors cause an invalidation for the cache line in the modified state. A cache line can be loaded in the exclusive if this is the only line requested in the same level. A cache line can be shared among other processors through L1 to L1 transfer, but the line can only be read. All other cache lines will be in Invalid state.
- MESIF [3] protocol adds on to MESI protocol by supporting a Forward (F) state which specifies the provider for a clean cache line which may be in a shared or exclusive state. This optimization reduces the number of redundant replies for a shared cache line.

- Other variants - MOESI protocol is a sister protocol to MESIF which introduces an owner state that specifies the owner of a cache line.

2.2 Directory-based protocols

Directory-based protocols require a centralized or decentralized data structure to keep track of holders of cache lines and can scale well with an increase in the number of cores. Directory-based protocols are the go-to when it comes to distributed nodes, where the cost of a naive broadcast would be too expensive due to communication over the network.

3 Approach

Our POPS-inspired [2] optimization for maintaining cache coherence uses additional hardware to support directory-based protocol's performance without the overhead for cache line modification to store the sharer bit vector or pointer to the next sharer. The following changes in the hardware are required to implement the proposed optimization

3.1 Modification to hardware

- Cache line - No changes are needed in the cache line, we can reliably use the same cache line as a MESI-supported cache would use.

- Bloom Filter Data Structure - Bloom filters are probabilistic data structures used to check set participation using hashing algorithms. Bloom filters guarantee no false negatives at the cost of false positives which serves the use case well as we can always check the cache after we detect it probably has the address stored. Another advantage of bloom filters is that they are space-efficient, the internal state can be stored in a bit vector of size 256 to have a false positive rate of less than 1 percent for 10000 addresses. In our bloom filter data structure, we allocate a line of 1024 bits to each processor and upon a local L1 miss, the bloom filter is used to retrieve the list of potential sharers of the address. The diagram below demonstrates the physical layout of the bloom filter data structure and the list of processors, indicated by the bit map, that potentially hold the address.

  As can be observed in the diagram below, the address's presence is checked for in each bloom filter and a bit vector is produced which indicates the probable presence or definite absence of that address in the cache. In the example below, Processors 0, 4 and 5 probably have the address and we don't need to broadcast to each cache. In this project, we used an existing bloom filter implementation [1] that provides us the flexibility to choose the number of bits in the internal bit map.

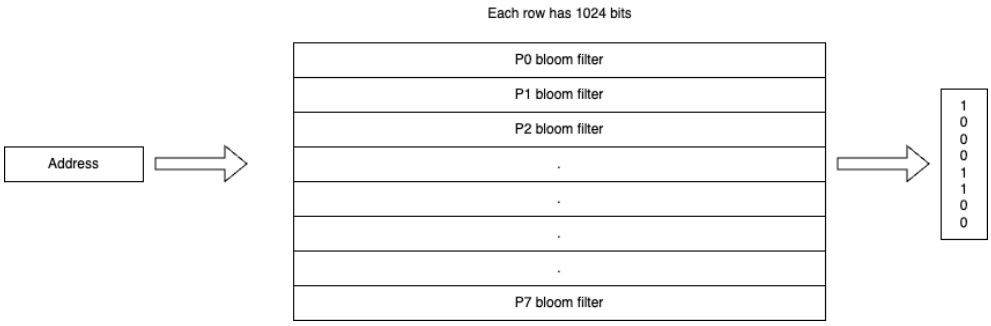

Figure 2: Proposed Bloom filter data structure

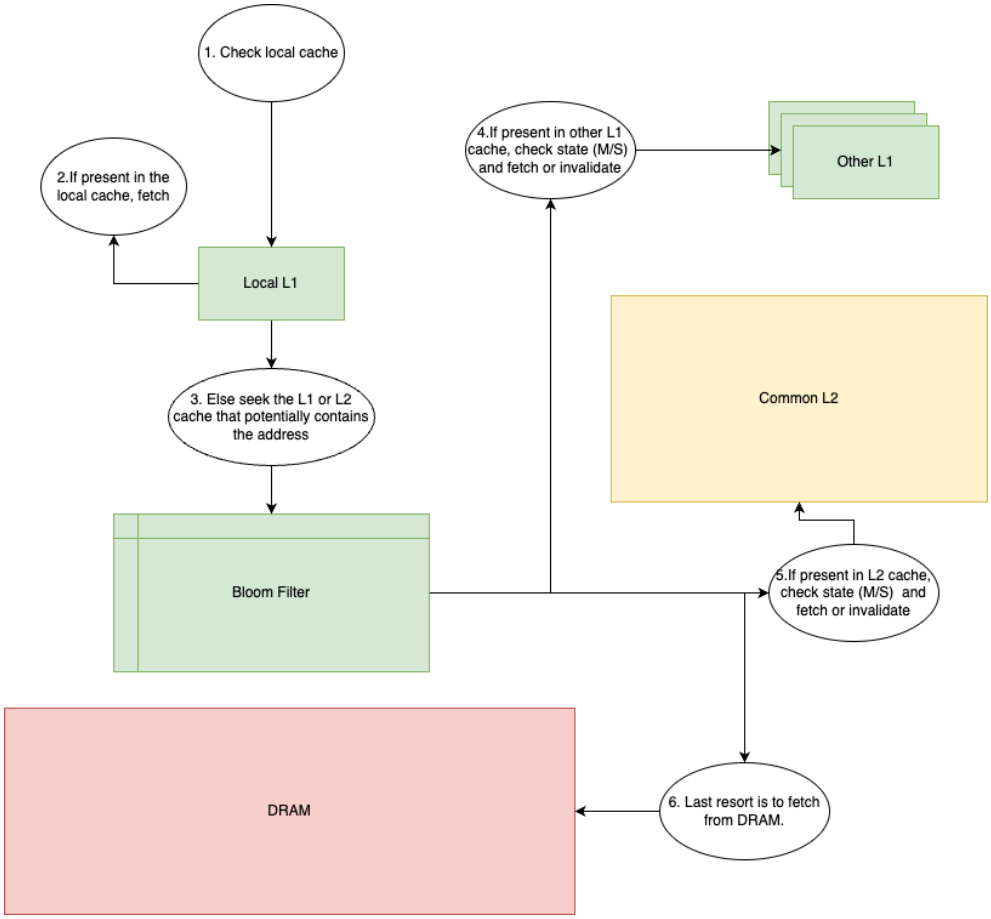

Figure 3: State flow diagram from POPS coherence optimization

3.2 State flow diagram for POPS

The section describes the behavior under different scenarios for POPS-inspired coherence protocol.

3.2.1 Loads

- When the address is present in the local cache, it can be read as such.
- When the address is present in another cache and is shared or in exclusive mode - Add the address to the processor's bloom filter and load it to the cache in the shared state.
- When the address is present in another cache and is in modified state - Invalidate the address in another cache, add the address to the processor's bloom filter and load it to cache in the exclusive state.
- When the address is present in the L2 cache - Add the address to the processor's bloom filter and load the address to local cache.
- Load from DRAM if not found in all of the above.

3.2.2 Stores

- When the address is present in the local cache, it can be written to as such.
- When the address is present in another cache and is shared, exclusive mode or modified - Invalidate the cache line in all the sharer caches, add the address to the bloom filter and load the address to the local cache.
- When the address is present in the L2 cache - Add the address to the processor's bloom filter and load the address to local cache.
- Load from DRAM if not found in all of the above.

3.3 Assumptions

- We intend to keep a globally accessible bloom filter data structure to keep track of addresses held by each cache rather than associating a pointer with each cache line to keep track of the sharers to minimize space usage.
- We assume that a secure hash instruction set can be used to compute address hashes efficiently to support the bloom filter data structure.
- We assume the cost of bloom filter lookup will be more than that of an L1 lookup by an order of magnitude.

4 Results

We generated the traces using Dtrace which were further processed to segregate the traces for individual processors. The charts below compare the cost of memory fetches from different levels in the memory for MESIF and POPS-inspired coherence protocol.

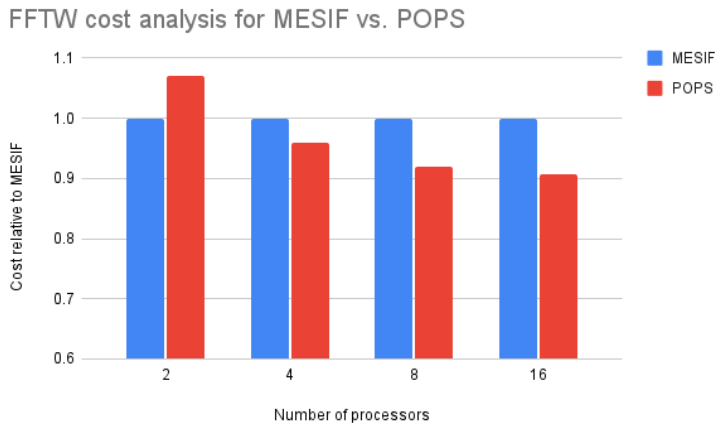

Figure 4: FFTW cost analysis

4.1 Analysis of FFTW

The traces used in the analysis for FFT were generated by FFTW library for 16k points under the multithreaded setting. The trends below show a slightly expensive cost of operation for 2 threads while the performance improved as the number of caches increased.

4.2 Analysis of Mandelbrot set

The traces used in the analysis for the Mandelbrot set were generated for 900x1100-sized image with dynamic scheduling. The trends below show a constant decrease in memory cost as the number of processors increase.

4.3 Analysis of Migratory and ProdCon

We developed and tested our initial implementations against the reference traces for migratory and prodcon traces. The results below demonstrate the decrease in memory cost when using the POPS-inspired coherence protocol.

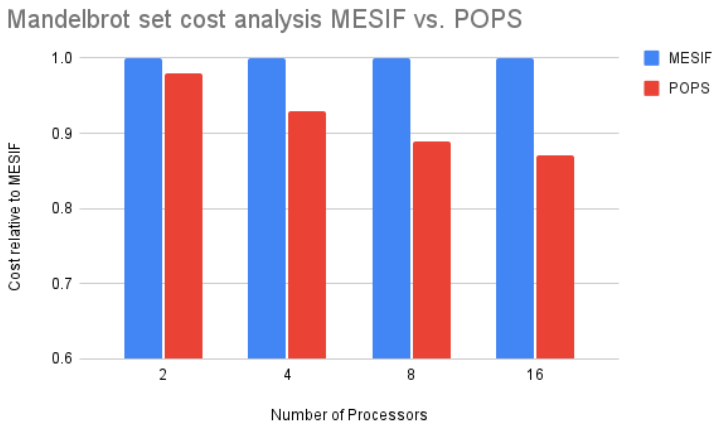

Figure 5: Mandelbrot set cost analysis

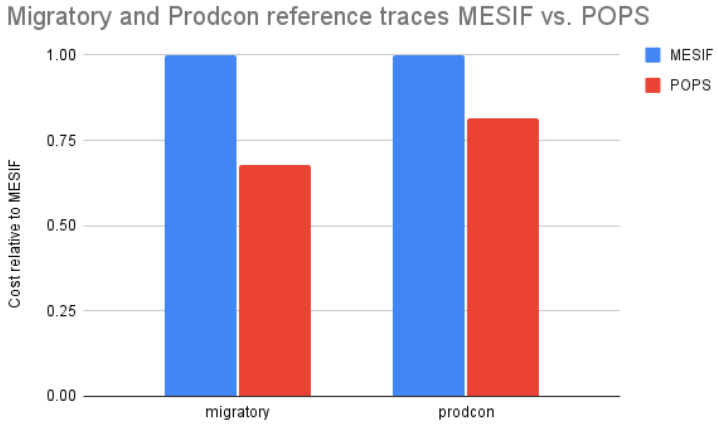

Figure 6: Migratory and prodcon trace cost analysis

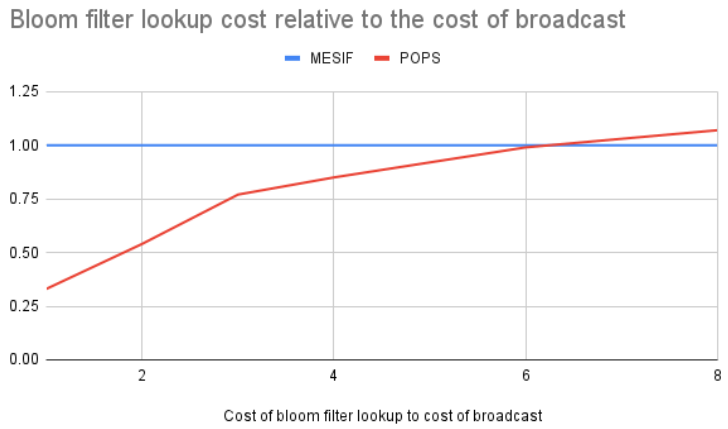

Figure 7: Ablation study for cost of bloom filter lookup

4.4 Ablation study for the cost of bloom filter lookup

To understand the behavior of bloom filter under different cost models, we perform the ablation study by varying the cost of Bloom filter lookup for FFT traces for 8 processors. It can be observed that as long the cost of a bloom filter lookup is less than 6x the cost of broadcast, our POPS-inspired coherence protocol will perform better than MESIF.

5 Conclusion

This sections concludes our findings about an adaptation for a coherence protocol that seeks to achieve a middle ground between snooping-based and directory-based coherence protocols. The results for standard workloads show that the performance for this variant improves as the number of processors increases. The experiment does not account for the effect of introducing a bloom filter component on the interconnect and the increase in energy consumption due to the lookups.

References

[1] *Arash Partow's bloom filter implementation*. `https://github.com/ArashPartow/bloom/tree/master`.

[2] Hemayet Hossain, Sandhya Dwarkadas, and Michael C. Huang. "POPS: Coherence Protocol Optimization for Both Private and Shared Data". In: *2011 International Conference on Parallel Architectures and Compilation Techniques*. 2011, pp. 45–55. DOI: `10.1109/PACT.2011.11`.

[3] *Intel's MESIF protocol for Haswell and above*. https://patents.google.com/patent/US6922756B2/en. 2008.