Implementing Optimized Cache Coherence Protocols in Multicore Systems

TEAM: Bharathi Sridhar (bsridha2), Tanvi Daga (tdaga)

URL: https://github.com/bharathi2203/418-Directory-Based-Cache-Coherence/tree/main

SUMMARY:

In this project, we are implementing and analyzing advanced cache coherence protocols for multicore systems. We will focus on the directory-based scheme, its variants like the limited pointer and sparse directory schemes, and their effects on system performance. Our goal is to optimize these protocols by adjusting parameters such as cache line size, associativity, and replacement policy, and then measuring critical performance metrics including read and write hits and misses, cache evictions, and interconnect traffic. These implementations aim to provide insights into achieving efficient data consistency and enhanced performance in multicore environments. We will perform a detailed analysis of the performance of each protocol across different traces on the GHC and PSC machines.

BACKGROUND:

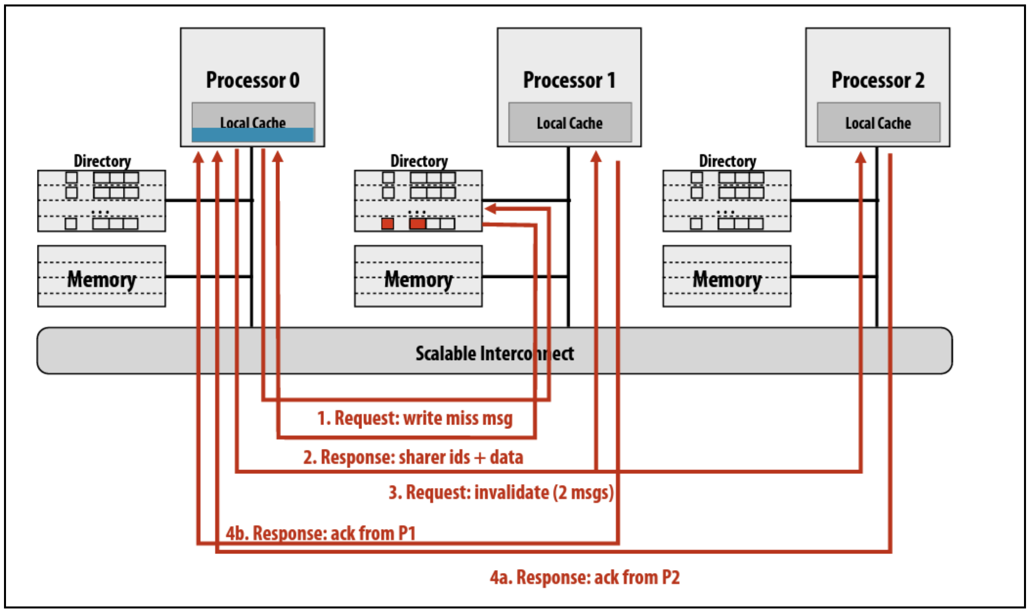
In multicore systems, dependencies are created when different cores often access and modify shared data. Managing these dependencies effectively to maintain data consistency and coherence without depreciating performance is a significant challenge. We need to ensure that when one core modifies data, other cores relying on this data are notified or updated efficiently. Additionally, the time taken to access memory can vary depending on the memory's location relative to a specific processor. This non-uniformity adds another layer of complexity to designing cache coherence protocols, as they must account for varying access times and their impact on overall system performance. A good cache coherence protocol must be robust and must be able to perform well against different applications that might have very different memory access patterns. Some may demonstrate high locality of reference, where repeated accesses to the same data occur, while others might have random access patterns.

One set of protocols discussed in lecture are the MSI, MESI, and MOESI snooping protocols which implement different strategies to maintain cache coherence by ensuring that multiple caches have consistent views of shared data.

Overall, snooping based cache coherence protocols are not scalable as they rely on broadcasting which generates a lot of interconnect traffic. However in directory based cache coherence, each processor tracks the state of a cache line in a directory entry and uses point-to-point messages between requesting and home node to maintain cache coherence. This helps avoid broadcasting.

For our project, we aim to implement a few other cache coherence protocols which weren't discussed in as much detail in class. These include:

Directory based scheme:


Processor 0
Local Cache
Processor 1
Local Cache
Processor 2
Local Cache
Directory
Directory
Directory
Memory
Memory
Memory
Scalable Interconnect
1. Request: write miss msg
2. Response: sharer ids + data
3. Request: invalidate (2 msgs)
4b. Response: ack from P1
4a. Response: ack from P2

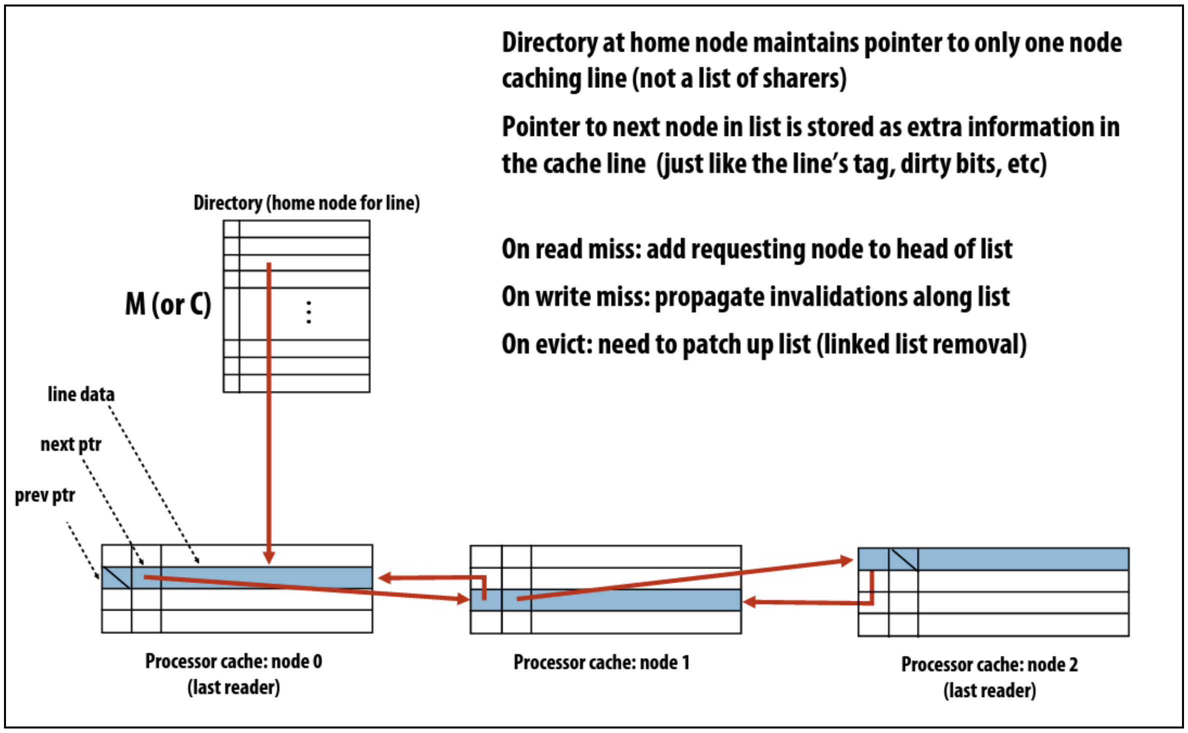
The directory-based cache coherence scheme maintains a centralized directory that tracks the state of each cache line in the system. Each directory entry indicates which processors have a copy of the cache line and whether it's been modified. A simple directory scheme has a large storage overhead, which limited pointer and sparse directory schemes can help reduce.

Limited pointer scheme:

The limited pointer scheme is a variation of the directory-based approach, designed to reduce storage overhead. In a limited pointer scheme, rather than having a bit per processor and entry per memory line, a limited number of pointers are stored per directory line. This effectively tracks cache line ownership while conserving space, but it can be less efficient in scenarios with high cache sharing.

Sparse directory:

Sparse directories further optimize the directory-based approach by aligning the directory size with the cache size. Each entry in this scheme stores the tag of the cache line, rather than information about every processor. This significantly reduces the storage requirement compared to full and limited pointer directories but may lead to increased cache misses and memory accesses in certain scenarios.


Directory at home node maintains pointer to only one node caching line (not a list of sharers)
Pointer to next node in list is stored as extra information in the cache line (just like the line's tag, dirty bits, etc)
On read miss: add requesting node to head of list
On write miss: propagate invalidations along list
On evict: need to patch up list (linked list removal)
Directory (home node for line)
M (or C)
line data
next ptr
prev ptr
Processor cache: node 0 (last reader)
Processor cache: node 1
Processor cache: node 2 (last reader)

We plan to examine/compare the performance of each method on system performance by altering parameters like cache line size, associativity, and replacement policy. Our analysis will involve measuring key metrics such as the number of read and write hits and misses, the frequency of cache evictions, and the amount of interconnect traffic etc.

CHALLENGES:

Parallelizing cache coherence is complex due to the intricate dependencies and memory access characteristics inherent in multicore systems. We aim to understand these challenges better and explore ways to optimize cache coherence.

Shared data among cores creates dependencies which require careful management to ensure consistency. Some applications may show high locality of reference, while others might exhibit more random access patterns. Different cores may execute different parts of a program or different programs entirely, leading to different paths and varying demands on the cache coherence system. Thus, the key challenges include implementing protocols accurately, creating effective computational traces, and understanding the relationship between memory access patterns and coherence protocol efficiency.

RESOURCES:

We have found 2 computer system simulator platforms that we can extend to add new coherence protocols: SST (https://sst-simulator.org/) and gem5 (https://www.gem5.org/). We can also extend the cache simulator from 15-213/18-213.

Sources:

CMU 15-418: Spring 2019: Lecture 13 -
https://www.cs.cmu.edu/afs/cs/academic/class/15418-s19/www/lectures/13_directory.pdf

GOALS AND DELIVERABLES:

PLAN TO ACHIEVE: Multicore cache simulator with directory, limited pointer, and sparse directory cache coherence schemes. Compare performance of different schemes implemented against a wide range of trace files.

HOPE TO ACHIEVE: Optimizing directory based coherence using intervention and request forwarding. Hybrid protocol, industry standard cache coherence protocol.

DEMO PLAN:

We will display comparative graphs and visualizations showing the performance of various protocols (including MSI, MESI, MOESI, directory-based, limited pointer, and sparse directory schemes) under different workload scenarios. These graphs will highlight performance achieved across the various metrics mentioned above such as cache hit/miss rates, evictions, and interconnect traffic. We will also create speedup graphs and efficiency data that compare the performance of our implementations against baseline models.

PLATFORM CHOICE:

We will use GHC machines for local development and testing, and then use the PSC machines so we can run traces on many cores. Using the PSC machines will also allow us to test the robustness of our implementations and provide a greater distribution of data for analysis.

SCHEDULE:

Week	**Task(s)**
Week 1: 11/15 - 11/19	- Initial Research and Planning: Define project goals, scope, and specific requirements. - Research existing cache coherence protocols and implementation details. - Choose appropriate simulation tools (like SST or gem5). - Collect various trace files with various work distributions.
Week 2: 11/20 - 11/26	- Develop the directory-based cache coherence scheme. - Extend the directory-based scheme to include the limited pointer approach. - Conduct initial tests to debug and verify the correct functioning of each implemented protocol.
Week 3: 11/27 - 12/3	- Continue/finish implementing limited pointer approach. - Further extend to implement the sparse directory scheme. - Refine and optimize the protocols based on initial analysis results. - Work on Milestone Report.
Week 4: 12/4 - 12/10	- Compare the optimized protocols against baseline models and each other. - Experiment with different cache line sizes, associativity levels, and replacement policies. - Perform advanced testing under high-load and diverse workload scenarios.
Week 5: 12/11 - 12/15	- Work on Project Final Report. - Prepare for Final Poster Session. - Buffer period for weeks 1 to 4 to finish up “plan to achieve” goals. - Use any extra time to work on “hope to achieve goals”.