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 OF PROGRESS SO FAR:

We design various data structures that we require to simulate a multiprocessor cache system. This included:

- Cache Line: Includes state (Invalid, Shared, Exclusive, Modified), tag, valid bit, dirty bit.
- Cache Set: Collection of cache lines.
- Cache: Contains multiple sets, statistics (hits, misses, evictions).
- Directory Entry: Contains a bit vector representing the presence of the block in each cache.
- Limited pointer scheme directory entry: Contains a vector storing the processor IDs where the cache line is present
- Directory: Array of directory entries.
- Message: For communication, includes type (Read Request, Write Request, Invalidate, etc.), sender, receiver, address.
- Interconnection Network: Handles message routing between processors and memory.

We then needed to initialize all the structs, and implemented basic single cache operations.

- Create and Initialize Caches: For each processor, create a cache with direct-mapped or set-associative configurations.
- Initialize Directory: Set up the central directory with a bit/processor ID vector for each memory block.
- Set Up Processors: Each processor should have a unique ID and associated cache.
- Read/Write Operations: Implement how caches handle read and write requests, including updating cache state and statistics.
- Cache Miss and Replacement: Define behavior for cache misses and evictions, including interacting with the directory. We decided to implement a Least Recently Used (LRU) policy.
- For the limited pointer approach, if the pointers overflow, we will kick out an old sharer.

We then implemented the code required to simulate the directory and the network which includes the interconnect and additional queues and associated functions to implement the message passing communication as a part of the protocol:

- Handling Requests: Implement how the directory responds to read and write requests from caches.

- Update and Invalidate: Implement logic for updating directory entries and sending invalidate or update messages to caches.
- Message Creation and Handling: Define how messages are created (e.g., on cache misses) and processed.
- Simulate Network: Implement a basic network to route messages between caches and the directory using an interconnect.

Test cases / trace files:
We also generated trace files for debugging/benchmarking purposes using a python script included in the traces folder. The script is capable of generating traces with various access patterns including a fully random access pattern, matrix multiplication, transpose, distributed sequential, interleaved and false sharing based access patterns.

CHALLENGES:

Once we had a basic directory, cache and interconnect structure implemented we began implementing the actual communication required for the different cache coherence protocols. We began by implementing a centralized directory based approach but realized that this implementation would not be very performant as the number of processors and memory modules increases, a central directory can become a bottleneck. Each memory access by any processor requires interaction with the central directory, which can limit the system's ability to scale. So we decided that despite distributed directories being more complex to implement and manage it would be a better design choice.

As a result, we are a bit behind schedule but we still believe that we will be able to complete our “plan to achieve” goals. Our updated goals and schedule below reflects the changes we plan to make.

UPDATED GOALS AND DELIVERABLES:

PLAN TO ACHIEVE:

- Multicore cache simulator with distributed directory, limited pointer, and sparse directory based cache coherence schemes.
- Compare performance of different schemes implemented against a wide range of trace files.

HOPE TO ACHIEVE:

- Multi-thread the sequential cache simulator implementations.
- Optimizing directory based coherence using intervention and request forwarding.

UPDATED DEMO PLAN:

We will display comparative graphs and visualizations showing the performance of various protocols (distributed 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.

SCHEDULE:

Half Week	Task(s)
12/3 - 12/6	- Complete implementing distributed directory based cache simulator. (Bharathi) - Complete implementing limited pointer based cache simulator. (Tanvi) - Generate trace files using pintool. (Bharathi, Tanvi)
12/6 - 12/9	- Complete implementation of sparse directory based cache simulator. (Bharathi, Tanvi)
12/10 - 12/13	- Debug & tune implementation to fix any bugs/ improve performance (Bharathi, Tanvi). - Work on benchmarking protocols, creating graphs, tables etc (Bharathi, Tanvi). - Work on “hope to achieve” goals. (Bharathi, Tanvi)
12/13 - 12/15	- Work on the final report write up. (Bharathi, Tanvi) - Work on the final report data, plots, tables and images. (Bharathi, Tanvi) - Work on poster slides. (Bharathi, Tanvi)