Adaptive Write-Update and Write-Invalidate Cache Coherence Protocols for Producer-Consumer Sharing

By Bangjie Liu, Hao Li

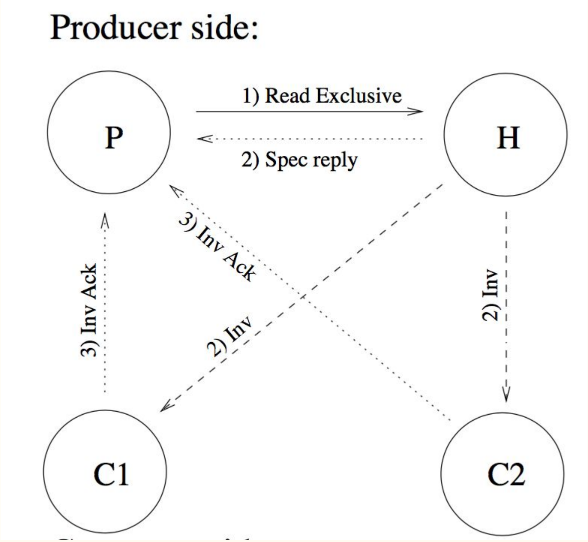
Producer-Consumer Sharing

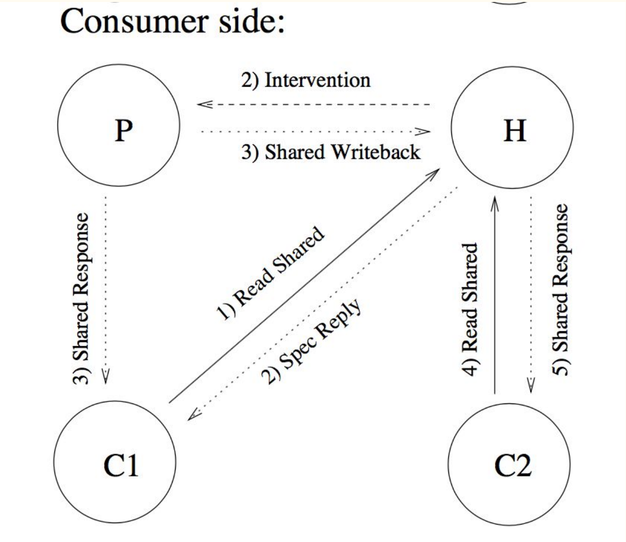
How to make it efficient from architectural perspective ?

Problem

Typical cc-NUMA systems use directory-based write-invalidate protocols[1], which is inefficient for producer-consumer sharing.

- Remote misses: modern processors don't support capability to push data to processor caches
- Communication traffics: 3 hops, write contention lead to ping-ponging





[1] Cheng, Liqun, John B. Carter, and Donglai Dai. "An adaptive cache coherence protocol optimized for producer-consumer sharing." *High Performance Computer Architecture, 2007. HPCA 2007. IEEE 13th International Symposium on*. IEEE, 2007. * This slide uses example figure 1 from [1]

Inspiration

Bus-based write-update protocol is efficient for producer-consumer sharing.

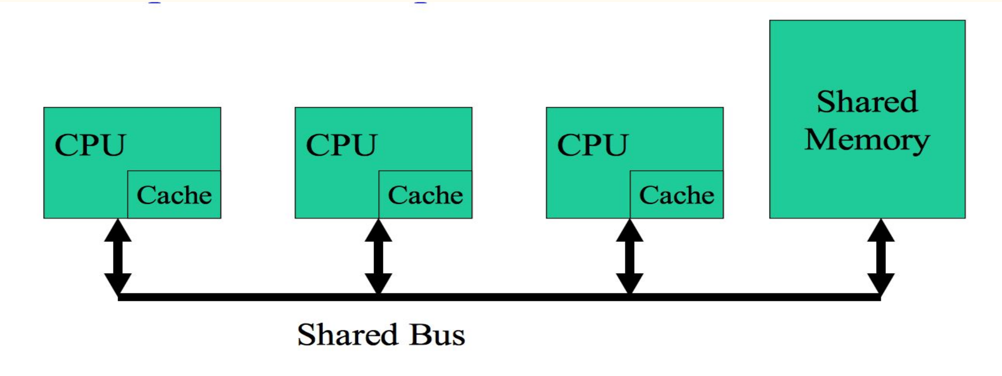
- Bus transactions are visible to all processors
- When local cache is updated by producer, the new data is broadcast to all consumer cache copies.

Benefits (for P-C sharing):

- Reduce communication traffics
- Reduce remote misses

Drawbacks (for general):

- Pattern dependent
- Bus bottleneck



[1] Prof.Nathan Beckmann, lecture slides “740: Computer Architecture Memory Consistency&Cache Coherence”, Carnegie Mellon University
[2] Sundararaman Nakshatra, Architecture(EECC551) slides. "Cache Coherence Protocol", http://meseec.ce.rit.edu/551-projects/fall2010/1-3.pdf
* This slide uses the example figure from adapted lecture slide by Ian Watson, “Cache coherence in shared-memory architectures”, University of Machester, http://www.cs.utexas.edu/~pingali/CS377P/2017sp/lectures/mesi.pdf

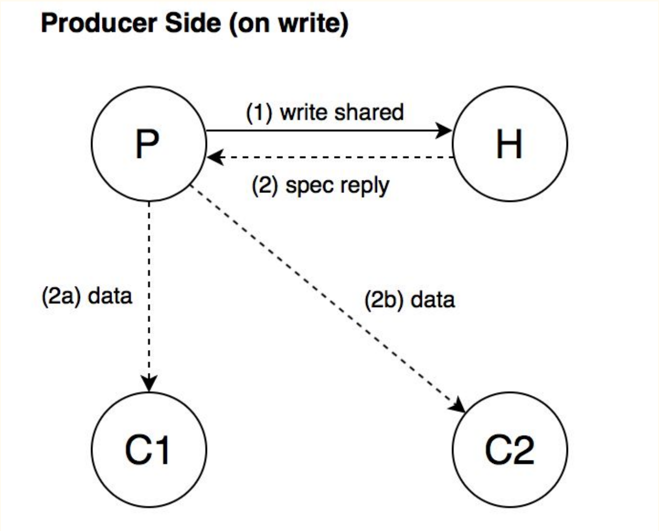
Can we do better?

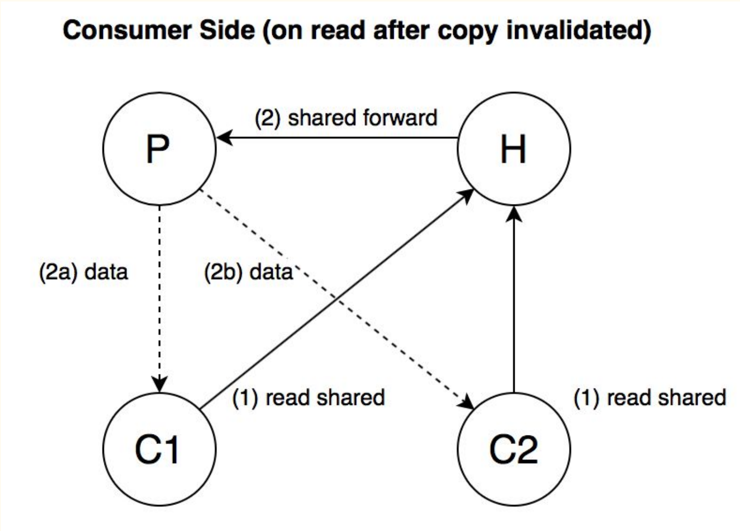
Adaptive Cache Protocol!

Solution

Directory-based adaptive cache coherence protocol

- Hardware support for cache → cache communication.
- Producer-Consumer sharing pattern detector.
- Capability to switch from write-invalidate to write-update.





Benefits

For P-C sharing:

- Reduce communication traffics: 3 hops → 2 hops
- Reduce remote misses: cache → cache

For general:

- Less bandwidth
- Only stakeholders retain copies

Potential Issues

- **Write propagation**
  - Hardware support, software test.

- **Write serialization**
  - Verify sequential consistency. (Murphi)

Goals

Goals

75%

- Implement a cache simulator and testing tools.
- Evaluate performance of directory-based cache-invalidate protocol on typical producer-consumer apps.

100%

- Implement proposed adaptive cache coherence protocols and producer-consumer sharing pattern detector.
- Evaluate the performance on producer-consumer apps.

125%

- Synchronize app threads to observe more accurate cache access events (eliminate pintool side effect).

Cache Simulator

Cache Simulator

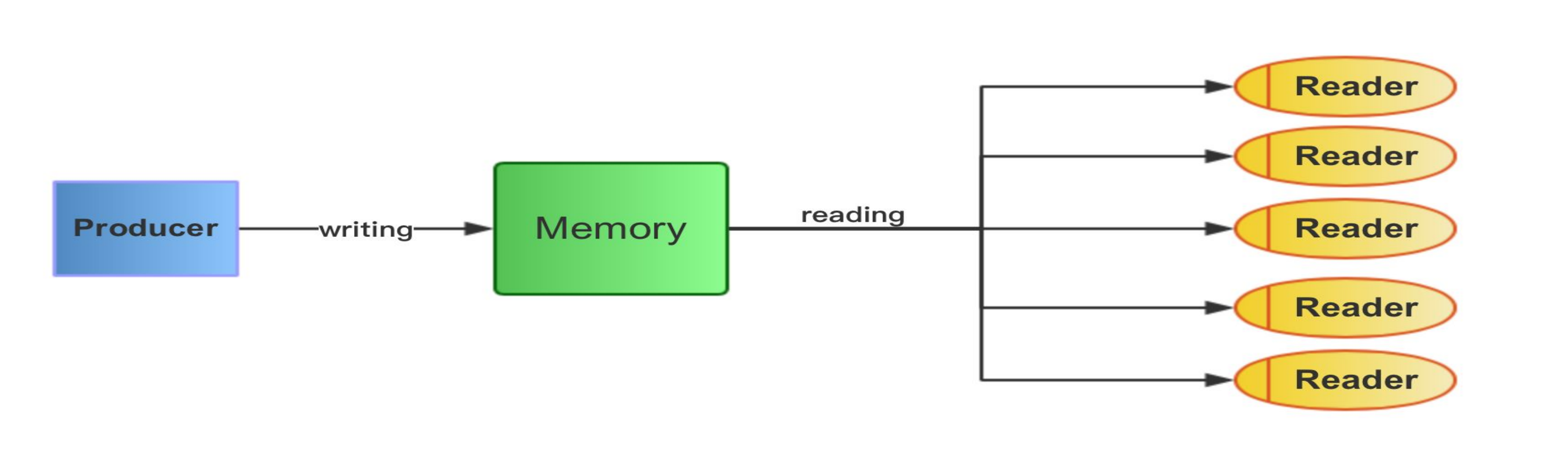
We now have a highly customizable cache simulator. (based on open source Dynamic Cache Simulator for Shared Address Space (SAS))

- MSI (modified, shared, invalid)
- Directory-based
- Write-back-allocate

Representative Application

Representative Application

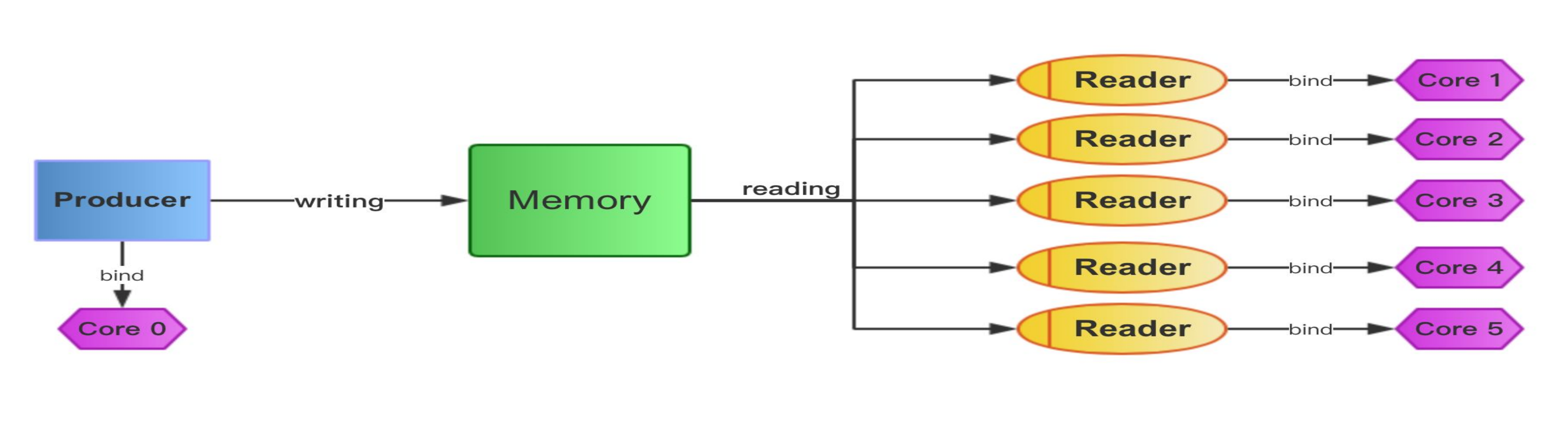
1. One producer keeps writing to a shared memory location while there are several readers keep reading from that particular location.
2. (#physical cores - 1) consumers



Representative Application

Potential issue: the OS can schedule threads among different cores, which will disturb cache.

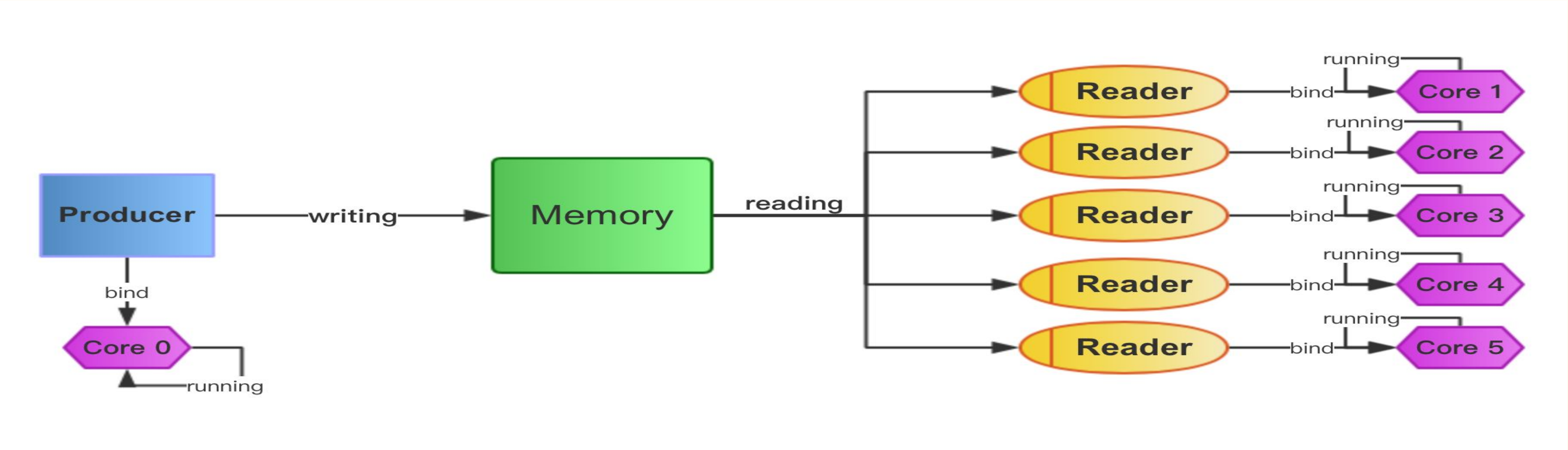
Solution: bind threads explicitly to their designated cores



Representative Application

Potential issue: binding could happen after threads start running because binding takes time.

Solution: each thread sleeps for a certain amount of time before executing.



Result Analysis

Result Analysis

Experiment environment:

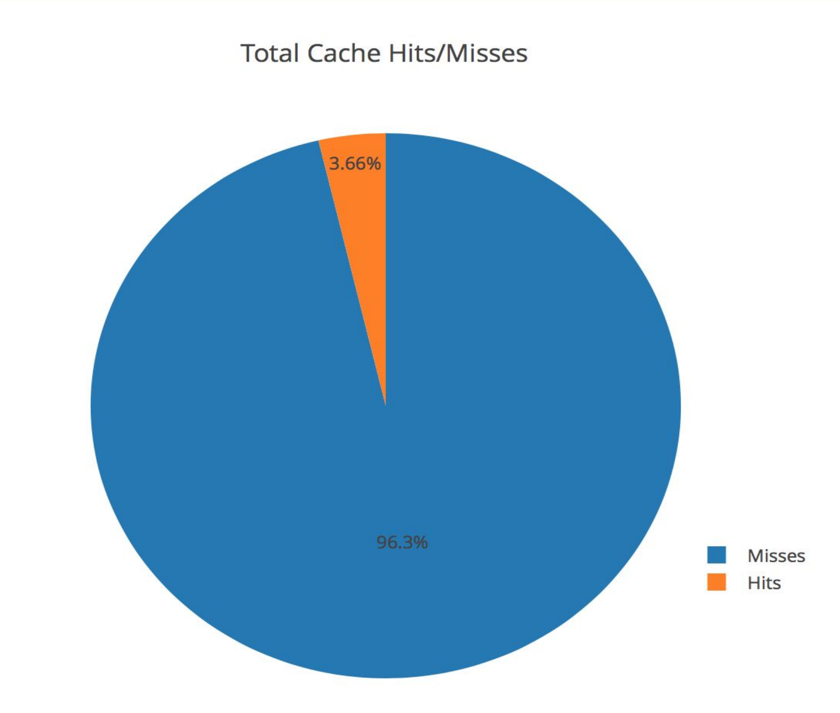
- GHC 46 <ghc46.ghc.andrew.cmu.edu>
- #Physical cores: 8
- L1 data cache: 32KB (#sets = 64, associativity = 8, line size = 64B)

Cache simulator:

- Only simulate L1 data cache
- Write strategy: WRITE_BACK_ALLOCATE
- Coherence protocol: MSI (modified, shared, invalid)
- Interconnect: Directory

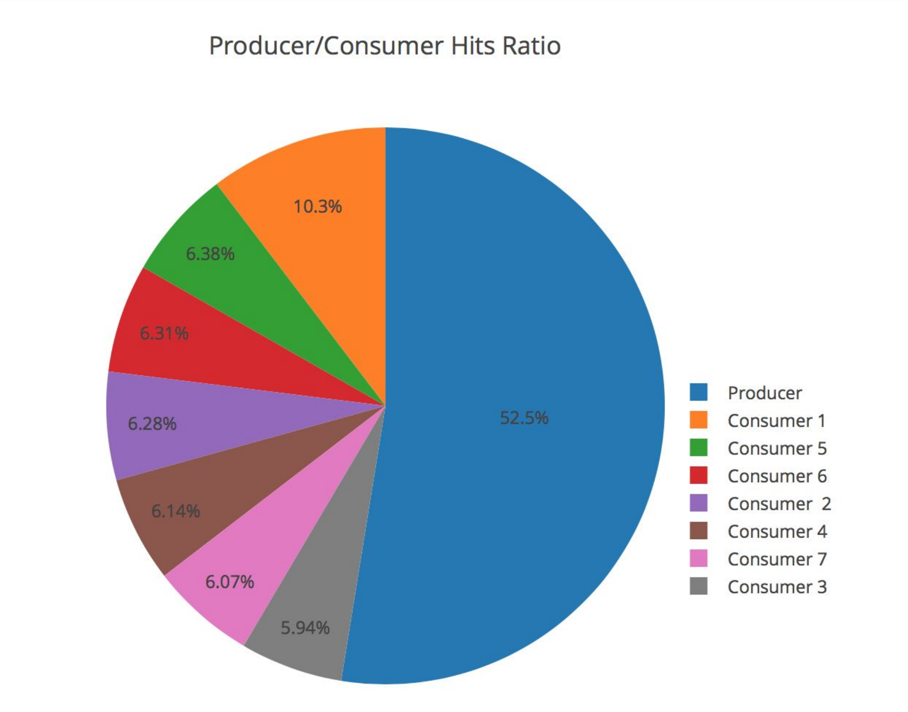
Result Analysis

- The overall performance is poor
- Cache hit rate is as low as 3.66%
- BECAUSE each write invalidates all copies of the data



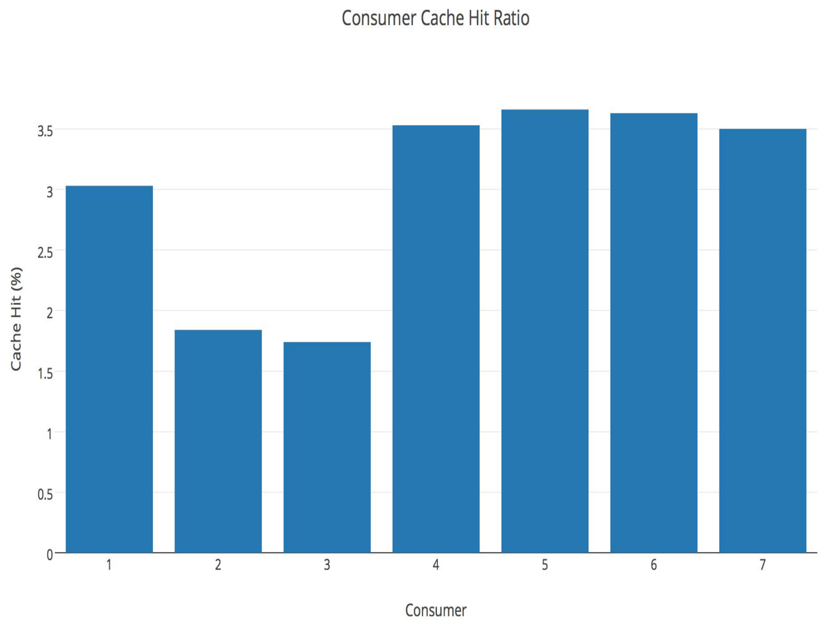
Result Analysis

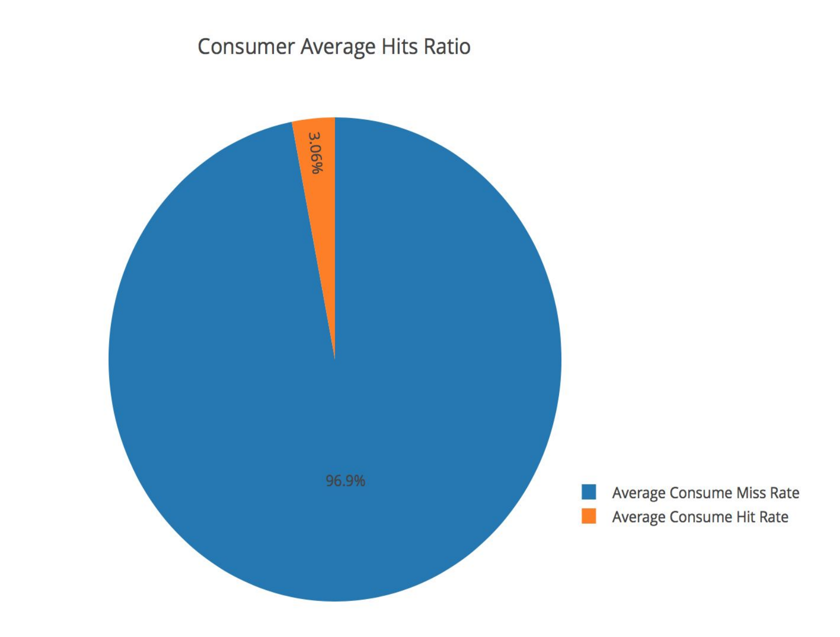
- Among all hits, the producer thread takes up more than 52%



Result Analysis

Consumer threads have extremely poor cache hit rates





75%

Plans

Plans

TODOs before moving on to 100% goal

1. Refactor cache simulator
2. Support more detailed analysis
   a. separate load/store hits and misses
   b. track directory misses
3. Fully test the cache simulator
   a. deal with unexpected output when the numbers of processors in cache simulator and application are different