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

Bangjie Liu
Language Technologies Institute
School of Computer Science
Carnegie Mellon University
bangjiel@andrew.cmu.edu

Hao Li
Language Technologies Institute
School of Computer Science
Carnegie Mellon University
haol2@andrew.cmu.edu

***Abstract*—Shared memory multicore systems play crucial roles in scientific and enterprise applications. They are efficient in general applications but perform poorly in applications that establish producer-consumer sharing patterns under write-invalidate cache coherence protocol. This is because write operations trigger the invalidation of copies reside in the cache of other cores, and thus introduces a large amount of memory reads that slow the system down. This project proposes an adaptive cache coherence protocol that can eliminate unnecessary memory accesses by speculatively pushing data modified by producer to potential consumers when producer-consumer sharing patterns are detected. We evaluate the proposed adaptive protocol on ... (results)**

I. Introduction

Shared memory multicore systems play increasingly important roles in both scientific world and the industry. Cache coherence protocol has great influence on performance of shared memory multicore systems. Producer-consumer sharing refers to situations in multi-process synchronization wherein multiple processes share a common, fixed-size buffer and some processes, as known as producers, keep writing new data to that shared buffer while some readers keep reading data from it [1]. The most popular cache coherence protocol used in modern multiprocessor architecture is directory-based write-invalidate protocol, which is inefficient for producer-consumer sharing due to extensive invalidation traffics and expensive remote misses. This project focuses on evaluating the performance of producer-consumer application under write-invalidate and write-update protocols and proposes an adaptive protocol optimized for producer-consumer sharing patterns.

Researches have been done in this area, but they focus more on eliminating unnecessary hops by using additional hardware for cache directory and the proposed pattern detector is not sophisticated enough [2]. Instead of mitigating remote misses, this project focuses on mitigate the slowest part — memory accesses, and proposes a sophisticated detector that investigates producer-consumer sharing patterns at a fine-grained level. More specifically, this project works on single-producer-multiple-consumer patterns and assumes that communications among caches on different cores are physically feasible.

The remainder of this report is organized as follows. We start in Section 2 with an overview of our goals and what we have accomplished. Section 3 and 4 describe the simulator in our study and our analysis of the performance of write-invalidate and write-update protocols on a representative producer-consumer application. In Section 5, we present our adaptive protocol, followed by a thorough evaluation on several applications in Section 6. Finally, we summarize our works and discuss future directions in Section 7 and 8.

II. Goals

[75%] Implement cache simulators and testing tools (stack trace, logger, etc.) to fully evaluate and analyze the performance of directory-based write-invalidate and write-update protocols on representative multi-threaded producer-consumer applications.

[100%] Implement proposed adaptive cache coherence protocols and producer-consumer sharing pattern detector. Evaluate and analyze the its performance on representative producer-consumer, as well as general-purpose applications.

[125%] Equip the cache simulator with functionalities that can synchronize application threads so as to observe more accurate cache access events (mitigate the effect of pintool instrumentation).

As we write this report, we have accomplished both 75% and 100% goals. And due to time limits, we decide to work on a more thorough analysis of the proposed protocol on multiple benchmarks instead of the 125% goal.

III. Cache Simulator

We implement a cache simulator and coherence protocols including write-invalidate, write-update, as well as the proposed adaptive versions of directory-based MSI (modified-shared-invalid) from scratch. The number of processors, cache sizes, number of sets, associativity, and line size are configurable. And it currently only simulates L1 data cache. In addition, it offers load/store hit rate statistics of the entire program, each core, and even all memory accessed. Besides, it supports cycle-accurate statistics by the cost estimation in Table I.

TABLE I: Cycle Cost Estimation

LOCAL CACHE ACCESS	3
REMOTE CACHE ACCESS	7
CACHE TO CACHE	4
MEMORY ACCESS	100

IV. Write-Invalidate and Write-Update Protocols Evaluation

In this section, we evaluate the performance, in terms of load/store hit rates, of write-invalidate and write-update protocols on a representative application that establishes producer-consumer sharing patterns. In the following evaluations, there are 1 producer and 2 consumers running on different cores. It is done on a Intel(R) Xeon(R) CPU E5-1660 v4 @ 3.20GHz machine with 32KB L1 data cache. And we only simulate L1 data cache.

A. Representative Application

In the application, there is a producer thread that keeps writing to a shared memory while there are multiple consumer threads reading for it as shown in Algorithm 1 and 2. A such self-defined application works well for evaluation purposes because it offers overall performance evaluation and more importantly a fine-grained investigation of shared memory behavior under different cache coherence protocols.

Algorithm 1 Producer Thread

```
1: data refers to a shared memory location
2: while True do
3:     GetLock()                    ▷ get global lock
4:     Produce(data)
```

Algorithm 2 Consumer Thread

```
1: data refers to a shared memory location
2: while True do
3:     GetLock()                    ▷ get global lock
4:     Consume(data)
```

B. Write-Invalidate Performance

Write-invalidate protocol works poorly on producer-consumer sharing patterns. As Figure 1 indicates, the overall load hit rate is acceptable. But for the shared data alone as shown in Figure 2, consumers' load hit rates are low due to a large amount of invalidation.

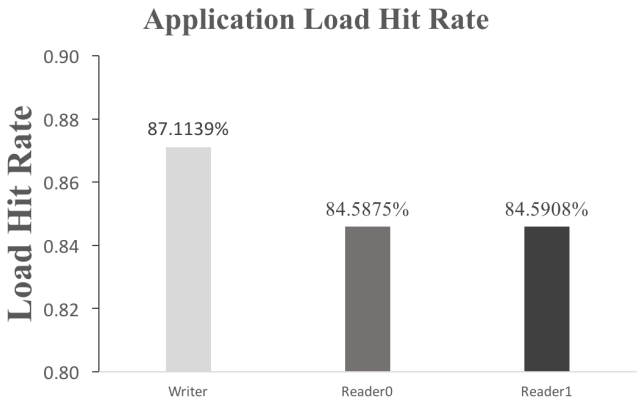

Fig. 1: Write-Invalidate Overall Load Hit Rates

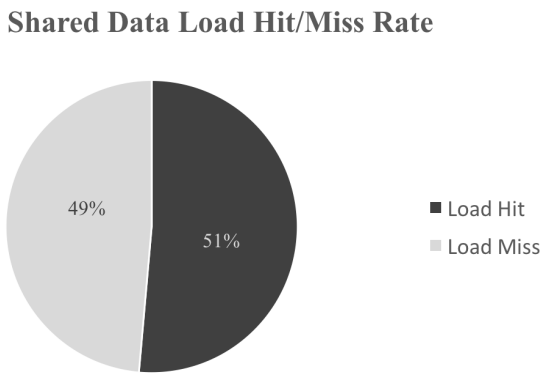

Fig. 2: Write-Invalidate Shared Data Load Hit Rates

C. Write-Update Performance

Write-update works well on producer-consumer sharing patterns. As Figure 3 and 4 shows, the overall performance is good and consumers have perfect load hit rates. This is because consumers now can get data via cache-to-cache communication without accessing memory.

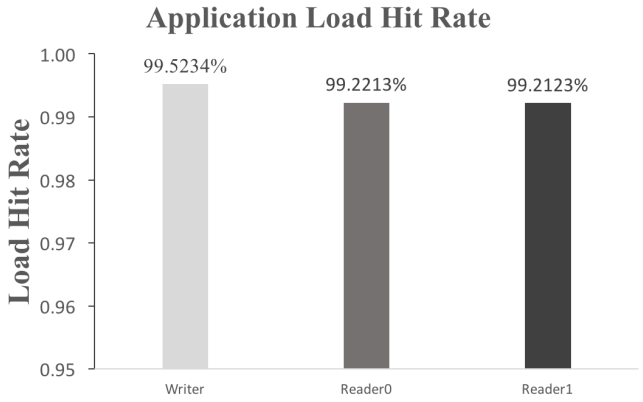

Fig. 3: Write-Invalidate Performance

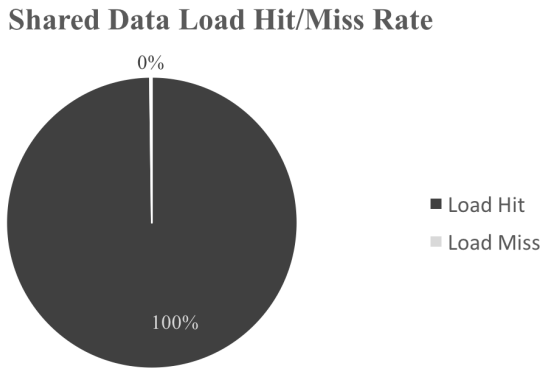

Fig. 4: Write-Invalidate Performance

V. Adaptive Cache Coherence Protocol

In this section, we introduce an adaptive cache coherence protocol that intelligently switches between write-invalidate and write-update protocols. It can mitigate unnecessary memory accesses by speculatively pushing data to consumers once producer-consumer patterns are detected. Additional bits are needed to track access history of each cache line for pattern identification. They are associated with directory lines which are extended to the structure shown in Figure 5.

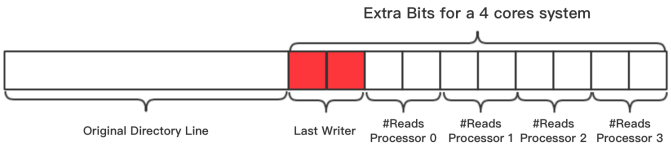

Fig. 5: Extended Directory Line Structure of a 4 Core Machine

The extra bits introduced for every cache line is

$$logN + 2 * N$$

where N is the number of processors, $last\_writer$ tracks the last one to write this cache line, and there are 2 saturating bits for every processor to track the number of read operations on this cache line. Cache accesses will trigger updates on the extra bits as shown in Algorithm 3 and 4. Note that the extra bits will be discarded to save space when its associated directory line is evicted.

Algorithm 3 On Write Operations

```
Let writer_id be the processor performing writes
if writer_id == last_writer then
    for all pid ≠ writer_id do                ▷ exclude itself
        if CountReads(pid) >= 1 then
            Cache2CacheDataPush(pid)          ▷ write-update
        else
            InvalidateDataCopy(pid)           ▷ write-invalidate
        SaturatingDecrease(pid)               ▷ decrease by 1
else
    for all pid ≠ writer_id do                ▷ exclude itself
        InvalidateDataCopy(pid)               ▷ write-invalidate
        ClearCount(pid)                       ▷ reset read counts
    UpdateLastWriter(writer_id)
```

Algorithm 4 On Read Operations

```
Let pid refer to a processor
for all pid do
    SaturatingIncrease(pid)                   ▷ increase by 1
```

VI. Adaptive Protocol Evaluation

In this section, we evaluate the performance of the proposed adaptive protocol on both general-purpose and producer-consumer sharing applications. In the following evaluations, there are 1 producer and 2 consumers running on different cores. It is done on a Intel(R) Xeon(R) CPU E5-1660 v4 @ 3.20GHz machine with 32KB L1 data cache. And we only simulate L1 data cache.

VII. Summary

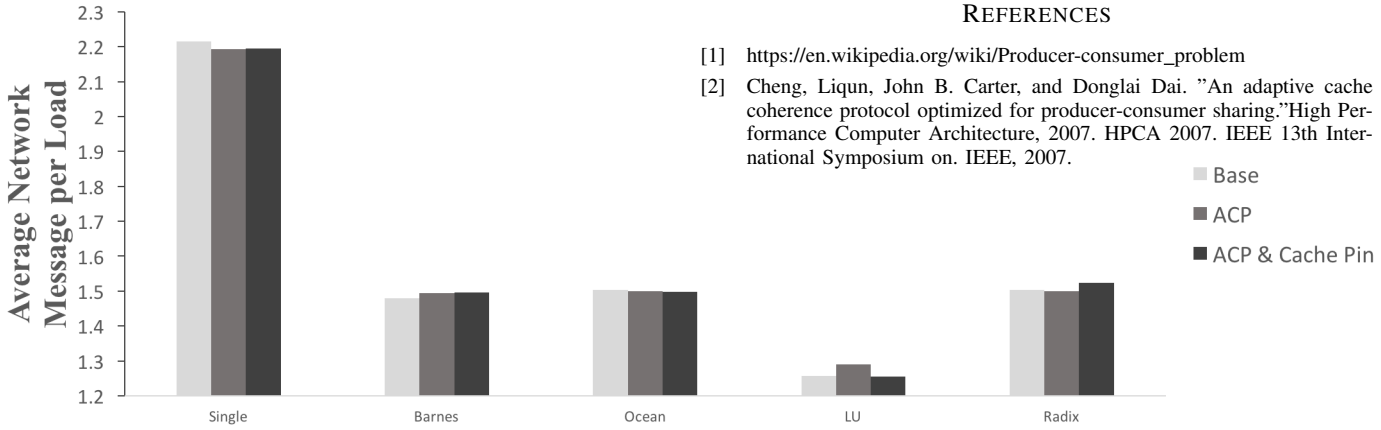

Fig. 6: Average Network Message per Load

VIII. Future Works

References

[1] https://en.wikipedia.org/wiki/Producer-consumer_problem

[2] 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.