A Quantitative Analysis of Shared LLC Pollution by Helper Threaded Data Prefetching on CMPs

Min Cai, Zhimin Gu

***Abstract*—Helper threaded data prefetching on chip multiprocessors (CMPs), in its most basic form, utilizes the processing power of underutilized neighboring cores that communicate via a shared last level cache (LLC) to improve the performance of a single-threaded program running in a CMP. However, the multiple memory reference/miss streams that come from the main thread (MT) and the helper thread (HT) can inevitably stress and pollute LLC if no effective LLC content management techniques are employed.**

In this paper, we present the methodology and mechanisms used in the quantitative analysis of the shared LLC pollution caused by helper-threaded data prefetching on CMPs, in the aim of providing insights on improving the effectiveness and timeliness of the scheme. Firstly, the simulation environment for simulating helper-threaded data prefetching on CMPs is described. Secondly, the methodology to classify and quantify the contribution of HT requests to the MT performance is discussed. Results of memory-intensive benchmarks from Olden and CPU2006 show that: (1) there is notable cache pollution caused by helper-threaded data prefetching on CMPs; (2) there is room for improving the effectiveness and timeliness of helper-threaded data prefetching on CMPs.

***Index Terms*—chip multiprocessors, helper threaded data prefetching, cache replacement, cache pollution**

I. Introduction

HELPER threaded data prefetching on chip multiprocessors (CMPs), in its most basic form, utilizes the processing power of underutilized neighboring cores that communicate via a shared last level cache (LLC) to improve the performance of a single-threaded program running in a CMP. However, the multiple memory reference/miss streams that come from the main thread (MT) and the helper thread (HT) can inevitably stress and pollute LLC if no effective LLC content management techniques are employed.

In this paper, we present the methodology and mechanisms used in the quantitative analysis of the shared LLC pollution caused by helper-threaded data prefetching on CMPs, in the aim of providing insights on improving the effectiveness and timeliness of the scheme. Firstly, the experimental framework for simulating helper-threaded data prefetching on CMPs is described. Secondly, the methodology to classify and quantify the contribution of HT requests to the MT performance is discussed. Results of memory-intensive benchmarks from Olden and CPU2006 show that: (1) there is notable cache pollution caused by helper-threaded data prefetching on CMPs; (2) there is room for improving the effectiveness and timeliness of helper-threaded data prefetching on CMPs.

Min Cai is with the School of Computer Science and Technology, Beijing Institute of Technology, Beijing, China, e-mail: min.cai.china@gmail.com.

Zhimin Gu is with the School of Computer Science and Technology, Beijing Institute of Technology, Beijing, China, e-mail: zmgu@x263.net.

The remaining of the paper is organized as follows. Section 2 introduces the simulation environment for simulating helper-threaded data prefetching on CMPs. Section 3 discusses the methodology used to classify and quantify the contribution of HT requests to the MT performance. Section 4 presents the results. Section 5 shows the previous work. Section 6 concludes the paper.

II. Simulation Environment

We use an in-house multicore architectural simulator named Archimulator in our experiments mentioned in this work. Archimulator is an execution-driven architectural simulator implemented in Java and running on Linux that supports application-only cycle-accurate modeling of multicore multi-threaded architectures. It has preliminary support of simulating Pthreads based parallel workloads.

III. Methodology

A. Overview

First, we need to track which LLC requests come from the main thread, and which LLC requests come from the helper thread. For example, consider a simulated multicore machine which has two cores where each core has two hardware threads. In our system call emulation mode simulation in Archimulator, without the OS intervention, one hardware thread can only run at least one software context (or called thread). Therefore, the typical software context to hardware thread assignments can be: core 0 thread 0 run context 0 (which is the main thread), core 0 thread 1 run context 1 (which is the pthread manager thread), core 1 thread 0 run context 2 (which is the helper thread), and core 1 thread 1 is idle. We use this configuration in the following discussions.

Second, to monitor the request and replacement activities in LLC, we need to consider the event when the LLC receives a request coming from the upper level cache (i.e., a hit or miss), which is named CoherentCacheServiceNonblockingRequestEvent in Archimulator. The event has a few important properties, e.g., the address of the requested LLC line, the requester memory hierarchy access, line found in the LLC, a boolean value indicating whether the request hits in the LLC, and a boolean value indicating whether the request needs to evict some LLC line. This event is similar to the comination of two events mentioned in [1].

Thirdly, in order to track the HT request state in the LLC and victims replaced by HT requests, we add two data structures to the LLC, namely htRequestStates and htRequestVictimCache. htRequestStates is a mirror cache of the LLC, which maintains an HT request state field for each

LLC line, which can be in three values: INVALID, MT and HT. htRequestVictimCache is an LRU cache that has the same structure of the LLC, but there is no mapping between each LLC line and each htRequestVictimCache line. Each htRequestVictimCache line contains a field, the type of field can be in three values: INVALID, NULL and DATA.

B. Algorithms

```
1  if(hitInLLC && requesterIsHT) {
2    totalHtRequests++;
3  }
4
5  if(requesterIsHT && hitInLLC && hasEviction) {
6    llc.setHT(set, llcLine.way);
7    victimCache.insertNullEntry(set);
8  }
9  else if(requesterIsHT && !hitInLLC && hasEviction &&
       !lineFoundIsHT) {
10   llc.setHT(set, llcLine.way);
11   victimCache.insertDataEntry(set, llcLine.tag);
12 }
13 else if(requesterIsHT && !hitInCache && hasEviction
     && lineFoundIsHT) {
14 }
15 else if(!requesterIsHT && !hitInCache && hasEviction
       && lineFoundIsHT) {
16   llc.setMT(set, llcLine.way);
17   victimCache.removeLRU(set);
18 }
19 else if(!requesterIsHT && !lineFoundIsHT) {
20   if(htRequestFound()) {
21     victimCache.removeLRU(set);
22     victimCache.insertDataEntry(set, llcLine.tag);
23   }
24 }
```

Figure 1. LLC Line Brought In

```
1  if(!mtHit && !htHit && vtHit) {
2    badHtRequests++;
3    victimCache.setLRU(set, vtLine.way);
4  }
5  else if(!mtHit && htHit && !vtHit) {
6    llc.setMT(set, llcLine.way);
7    goodHtRequests++;
8    removeLRU(set);
9  }
10 else if(!mtHit && htHit && vtHit) {
11   llc.setMT(set, llcLine.way);
12   victimCache.setLRU(set, vtLine.way);
13   victimCache.removeLRU(set);
14 }
15 else if(mtHit && !htHit && vtHit) {
16   victimCache.setLRU(set, vtLine.way);
17 }
```

Figure 2. Servicing MT and HT Requests

IV. Results

- mst 1000, HT version, HT params: ?, ?, ? , detailed simulation vs checkpointed simulation
- mst 10000, HT version, HT params: ?, ?, ? , checkpointed simulation

V. Previous Work

VI. Conclusion

TODO: our work: what? how? result? Further work?

Acknowlegments

This work was supported by the National Natural Science Foundation of China under the contract No. 61070029.

References

[1] B. Mehta, D. Vantrease, and L. Yen, “Cache showdown: The good, bad and ugly,” 2004. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.115.6866; http://www.cs.wisc.edu/~bsmehta/757/paper.pdf