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.

We assume here a two level cache hierarchy where L1 caches are private and the L2 cache is shared among all processor cores on a single chip.

The main contributions of this paper can be summarized as follows: TODOs.

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.

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

Simulator: we use an in-house multicore architectural simulator named Archimulator in our experiments mentioned in this work. Archimulator is an execution-driven architectural simulator written 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.

As shown in Tab.I, we simulate a target machine with 4-wide OoO 2 x 2 CMP, shared 4MB 8-way set-associative L2, private 32KB 8-way set-associative L1 data caches and 32KB 4-way set-associative L1 instruction caches, MESI coherence between L1s, switch based point-to-point on-chip interconnect.

Pipeline	superscalar width is 4; physical register file capacity: 128; decode buffer capacity: 96; reorder buffer capacity: 96; load store queue capacity: 48				
Branch Predictor	Perfect branch predictor (for the moment)				
Execution Units	Name	Count	Operation Lat.	Issue Lat.	
	Int ALU	8	2	1	
	Int Mult	2	3	1	
	Int Div		20	19	
	Fp Add	8	4	1	
	Fp Compare		4	1	
	Fp Convert		4	1	
	Fp Mult	2	8	1	
	Fp Div		40	20	
	Fp Sqrt		80	40	
	Read Port	4	1	1	
	Write Port		1	1	
Cache Geos	Name	Size	Assoc.	Line Size	Hit Lat.
	l1i	32KB	4	64B	1
	l1d	32KB	8	64B	1
	l2	4096KB	8	64B	10
Interconnect	switch based P2P topology, 32B link width				
Main Memory	4GB, 200-cycle fixed latency				

Table I
Baseline Hardware Configurations

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

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

III. Methodology

A. Definitions

Based on their contribution to the MT performance, on the LLC which is shared by MT, HT and some other threads, the following taxonomy of HT requests can be obtained:

- # Good HT requests = # HT requests that hit by MT before evicted;
- # Bad HT requests = # HT requests whose requested data are used later (or not used at all) by MT than the evicted data;
- # Ugly HT requests = # HT requests - # good HT requests - # bad HT requests.

B. Hardware Changes

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 application-only simulation using 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, which is shown in Fig.1. We use this configuration in the following discussions.

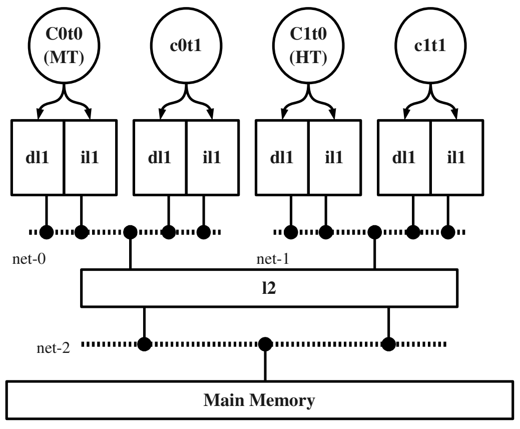

Figure 1. Simulated Multicore Architectures

Second, to monitor the request and replacement activities in LLC, we need to consider the event indicating the LLC receives a request coming from the upper level cache, whether it is a hit or a miss. 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 discussion in [1].

Thirdly, in order to track the HT request states in the LLC, we need to add one field to each LLC line to indicate whether the line is brought by the main thread (MT) or the helper thread (HT) or otherwise invalid (INVALID).

Fourthly, in order to track victims replaced by HT requests, we need to add an LRU cache named HT Request Victim Cache (HTRVC) to maintain the LLC lines that are evicted by HT requests. Similar to the evict table used in [1], the HTRVC has the same structure of the LLC, but there is no direct mapping between LLC lines and HTRVC lines. HTRVC has only the purpose of profiling, so it has no impact on performance.

C. Invariants

Similar to the two invariants proposed in [1], there are two invariants that should be maintained:

1) # of HT Lines in the LLC Set = # of Victim Entries in the HTRVC Set;
2) # of Victim Entries in HTRVC Set + # of Valid MT LLC Lines in Set $\leq$ LLC Set Associativity.

From the above two invariants, we can conclude that: # of HT Lines in the LLC Set+ # of Valid MT LLC Lines in Set $\leq$ LLC Set Associativity.

D. Actions taken on LLC and HT Request Victim Cache

Similar to the approach proposed in [1], corresponding actions should be taken in LLC and HTRVC when filling an LLC line (Fig.2) or servicing an incoming LLC request (Fig.3).

TODOs: dispatching and handling of CoherentCacheServiceNonblockingRequestEvent: dispatching in CoherentCache.AbstractFindAndLockProcess.doLockingProcess(..) handling in LastLevelCacheHtRequestCachePollutionProfilingCapability.serviceRequest(..)

IV. Preliminary Results

1) mst 1000, HT version (params: LOOKAHEAD=20, STRIDE=10), detailed simulation vs checkpointed simulation
   - detailed simulation
     - llc.totalHtRequests=584655
     - llc.goodHtRequests=584579
     - llc.badHtRequests=0
     - llc.uglyHtRequests=76
   - checkpointed simulation
     - llc.totalHtRequests=585150
     - llc.goodHtRequests=585079
     - llc.badHtRequests=0
     - llc.uglyHtRequests=71
2) mst 10000, HT version, checkpointed simulation
   - pending...
3) em3d 1000, HT version (params: LOOKAHEAD=20), detailed simulation
   - llc.totalHtRequests=539
   - llc.goodHtRequests=517
   - llc.badHtRequests=0
   - llc.uglyHtRequests=22
4) em3d 400000, HT version, checkpointed simulation
   - pending...

```
//Inputs:
//  hitInLLC: whether the request hits in LLC or not
//  requesterIsHT: whether the request comes from HT or not
//  hasEviction: whether the request needs to evict some
      data
//  lineFoundIsHT: whether the LLC line found is brought by
       HT or not
//  htRequestFound(): whether there is at least one line in
       the LLC set that is brought by HT

//HT miss
if(!hitInLLC && requesterIsHT) {
  totalHtRequests++;
}

//Case 1: HT evicts INVALID
if(requesterIsHT && !hitInLLC && !hasEviction) {
  llc.setHT(set, llcLine.way);
  htrvc.insertNullEntry(set);
}
//Case 2: HT evicts MT
else if(requesterIsHT && !hitInLLC && hasEviction && !
      lineFoundIsHT) {
  llc.setHT(set, llcLine.way);
  htrvc.insertDataEntry(set, llcLine.tag);
}
//Case 3: HT evicts HT
else if(requesterIsHT && !hitInLLC && hasEviction &&
      lineFoundIsHT) {
}
//Case 4: MT evicts HT
else if(!requesterIsHT && !hitInLLC && hasEviction &&
      lineFoundIsHT) {
  llc.setMT(set, llcLine.way);
  htrvc.removeLRU(set);
}
//Case 5: MT evicts MT, and there exists one HT line
else if(!requesterIsHT && !lineFoundIsHT) {
  if(htRequestFound()) {
    htrvc.removeLRU(set);
    htrvc.insertDataEntry(set, llcLine.tag);
  }
}
```

Figure 2. Actions Taken When Fillling an LLC Line

```
//Inputs:
//  mtHit: whether the request comes from MT and hits in
      the LLC
//  htHit: whether the request comes from HT and hits in
      the LLC
//  vtHit: whether the request comes from MT and hits in
      the HTRVC

//Case 1: VT hit => Good HT request (HT request evicted
      useful data)
if(!mtHit && !htHit && vtHit) {
  badHtRequests++;
  htrvc.setLRU(set, vtLine.way);
}
//Case 2: HT hit => Bad HT request (HT requested data hit
      before evicted data)
else if(!mtHit && htHit && !vtHit) {
  llc.setMT(set, llcLine.way);
  goodHtRequests++;
  htrvc.removeLRU(set);
}
//Case 3: HT and VT hit (Useful data evicted and requested
      back in by HT)
else if(!mtHit && htHit && vtHit) {
  llc.setMT(set, llcLine.way);
  htrvc.setLRU(set, vtLine.way);
  htrvc.removeLRU(set);
}
//Case 4: MT and VT hit(Useful data evicted and requested
      back in by HT and hit to by MT)
else if(mtHit && !htHit && vtHit) {
  htrvc.setLRU(set, vtLine.way);
}
```

Figure 3. Actions Taken When Servicing an LLC Request

V. Previous Work

TODO: previous work on prefetch taxonomies. Traditional coverage and accuracy metrics for h/w prefetches. -> good, bad and ugly breakdown of h/w prefetches. -> more fine-grained breakdowns of h/w prefetches. any previous work on cache request breakdowns for helper threaded data prefetching?

VI. Conclusion

TODOs: 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