A Taxonomy of HT Requests in 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 on a CMP. 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. As a first step, appropriate metrics should be used to measure the effectiveness and cache pollution of the HT scheme. However, due to the shared LLC structure, the traditional metrics of accuracy and coverage and good/bad classification for hardware prefetching cannnot be applied to describe the inter-thread interference and cache pollution in the helper threaded data prefetching.**

In this paper, based on the idea of cache pollution and HT request lateness, we present a taxonomy of HT requests to evaluate the effectiveness of the helper-threaded data prefetching on CMPs, in the aim of providing insights on designing and implementing effective LLC content management mechanisms for the HT scheme. Firstly, the taxonomy is proposed. Secondly, the hardware changes to support the taxonomy and algorithms for tracking HT requests and victims. Lastly, 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)[1], 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.

Several metrics have been proposed in the past for evaluating the effectiveness of hardware prefetching, among which prefetch accuracy and coverage are the most intuitive ones [2]. We can adapt the definitions of accuracy and coverage for hardware prefetching to the HT scheme. HT request accuracy is defined as the ratio of the number of useful HT requests to the number of total HT requests. And HT request coverage is defined as the ratio of the number of useful HT requests to the number of MT misses plus MT hits to HT requested data. Here, an HT request is called useful when its requested data is referenced by MT before evicted. Furthermore, similar to the approach to hardware prefetching, to measure coverage and accuracy, all HT requests can be categorized into “Useful” and “Useless” HT requests [2]. a “Useful” HT request is one whose brought data is hit by a MT request before it is replaced, while a “Useless” Ht request is one whose brought data is replaced before it is hit by a MT request.

However the above accuracy and coverage metrics for HT requests and the classification of “Useful” and “Useless” HT requests don’t care about cache pollution and HT request lateness, which are two kinds of deficiencies in the HT scheme.

In this paper, based on the idea of cache pollution and HT request lateness, we present a taxonomy of HT requests to evaluate the effectiveness of the helper-threaded data prefetching on CMPs, in the aim of providing insights on designing and implementing effective LLC content management mechanisms for the HT 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:

1) Proposes the cycle-accurate multicore simulation framework to evaluate the cache pollution effects and lateness for HT requests in the helper thread scheme for CMPs.
2) Presents the cache pollution aware classification of “good”, “bad” and “ugly” HT requests.
3) Presents the lateness measurement of HT requests which is a good indicator for potential improvement of the HT scheme.

Firstly, the taxonomy is proposed. Secondly, the hardware changes to support the taxonomy and algorithms for tracking HT requests and victims. Lastly, 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 presents a taxonomy of HT requests to assist evaluating shared LLC pollution caused by helper-threaded data prefetching on CMPs, the hardware changes to support the taxonomy and algorithms for tracking HT requests and victims. Section 3 presents the results. Section 4 shows the previous work. Section 5 concludes the paper.

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.

II. Methodology

A. A Taxonomy of HT requests

As illustrated in Tab.I, We propose a taxonomy of HT requests based on the cache pollution and HT request lateness. Among these types of HT requests, good HT requests have positive impact on MT performance. Ugly HT requests have little performance impact on MT performance because the requested data are not referenced by MT before evicted and they do not evict any data that will be used by MT. Bad HT requests are harmful for MT performance which should be prevented as much as possible. Late HT requests are good indicators for potential performance improvement of the HT scheme because late HT requests may be converted to good HT requests by finetuning the parameters of the HT scheme.

Table I
HT Request Taxonomy

Name	Description
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	Requests that are not good or bad
Late HT requests	In-flight HT requests that hit by MT requests

B. Hardware Support

1) 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 written in Java and running on Linux that supports application-only cycle-accurate modeling of multicore multithreaded architectures. It has support of simulating Pthreads based parallel workloads.

As shown in Tab.II, we simulate a target machine with 4-wide out-of-order CMP which has two cores where each core can support two hardware threads. It has 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.

2) Software Context to Hardware Thread Mapping: As shown in Fig.1, for a typical Pthreads based HT program, there are three threads when running: MT, HT and the Pthreads manager thread. The Pthreads manager thread takes the role of spawning, pausing and resuming HT by passing signals to HT. Consider a simulated target multicore machine which has two cores where each core supports 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 simply called thread). Therefore, the typical software context to hardware thread mappings can be: C0T0 -> MT, C0T1 -> Pthreads manager thread, C1T0 -> HT (C = core, T = thread). We use this context mapping in the following discussions.

3) Hardware Changes:

Pipeline	4-wide superscalar OoO 2 x 2 CMP; physical register file capacity: 128; decode buffer capacity: 96; reorder buffer capacity: 96; load store queue capacity: 48				
Branch Predictors	Perfect branch predictor				
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 Geometries	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 II
Baseline Hardware Configurations

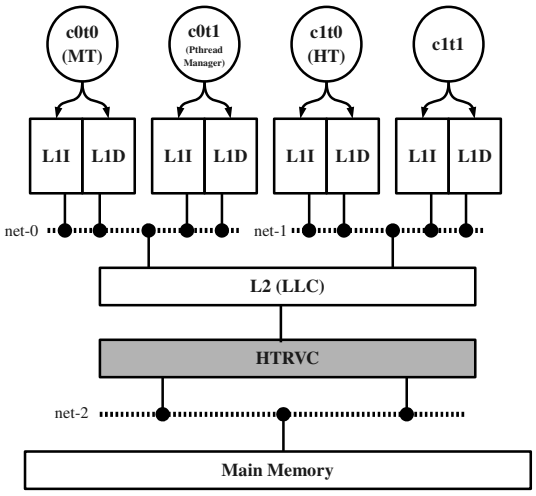

Figure 1. Simulated Multicore Architectures

a) LLC request and replacement event tracking: 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, whether it is a hit or a miss. The event has a few important properties to be used in the experiment, 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 one used in [3].

b) LLC HT request state tracking: 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).

c) LLC HT request victim state tracking: 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 [3], 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.

d) Detecting Late HT Requests: Lastly, in order to measure late HT requests, we only need to identify the event when an MT request hits to an LLC line which is being brought by an inflight HT request coming from the upper level cache. This can be accomplished by monitoring the LLC MSHRs or similar hardware components.

C. Algorithms for Tracking HT Requests and Victims

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.

HT lines refer to the cache lines that are brought by HT requests. From the above two invariants, we can easily conclude that: # of HT Lines in the LLC Set+ # of Valid MT LLC Lines in Set $\leq$ LLC Set Associativity. Actions should be taken in LLC and HTRVC when filling an LLC line or servicing an incoming LLC request.

1) Actions taken on Filling an LLC Line: When filling an LLC line, we should consider five cases:

1) An HT request evicts an INVALID line. In this case, no eviction is needed.
2) An HT request evicts an LLC line which is previously brought by an MT request. In this case, eviction is needed to make room for the incoming HT request.
3) An HT request evicts an LLC line which is previously brought by an HT request. In this case, eviction is needed to make room for the incoming HT request.
4) An MT request evicts an LLC line which is previously brought by an HT request. In this case, eviction is needed to make room for the incoming MT request.
5) An MT request evicts an LLC line which is previously brought by an MT request, and there exists one line that is brought by an HT request. In this case, eviction is needed to make room for the incoming MT request.

Specific actions taken on the above five cases are listed in Fig.2, where 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, and htRequestFound(): whether there is at least one line in the LLC set that is brought by HT.

2) Actions taken on Servicing an Incoming LLC Request: When servicing an incoming LLC request, either hit or miss, we should consider four cases:

1) Victim hit, which indicates a good HT request. This happens when HT request evicts useful data.
2) HT hit, which indicates a bad HT request. This happens when HT requested data is hit by MT request before evicted data.

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

//Case 1
if(requesterIsHT && !hitInLLC && !hasEviction) {
  llc.setHT(set, llcLine.way);
  htrvc.insertNullEntry(set);
}
//Case 2
else if(requesterIsHT && !hitInLLC && hasEviction && !
     lineFoundIsHT) {
  llc.setHT(set, llcLine.way);
  htrvc.insertDataEntry(set, llcLine.tag);
}
//Case 3
else if(requesterIsHT && !hitInLLC && hasEviction &&
     lineFoundIsHT) {
}
//Case 4
else if(!requesterIsHT && !hitInLLC && hasEviction &&
     lineFoundIsHT) {
  llc.setMT(set, llcLine.way);
  htrvc.removeLRU(set);
}
//Case 5
else if(!requesterIsHT && !lineFoundIsHT) {
  if(htRequestFound()) {
    htrvc.removeLRU(set);
    htrvc.insertDataEntry(set, llcLine.tag);
  }
}
```

Figure 2. Actions Taken When Fillling an LLC Line

3) HT and Victim hit. This happens when useful data is evicted and brought back in by HT request.
4) MT and Victim hit. This happens when useful data is evicted and brought back in by HT request and hit to by MT request.

Specific actions taken on the above five cases are listed in Fig.3, where mtHit: whether the request comes from MT and hits in the LLC, htHit: whether the request comes from HT and hits in the LLC, and vtHit: whether the request comes from MT and hits in the HTRVC.

```
//Case 1
if(!mtHit && !htHit && vtHit) {
  badHtRequests++;
  htrvc.setLRU(set, vtLine.way);
}
//Case 2
else if(!mtHit && htHit && !vtHit) {
  llc.setMT(set, llcLine.way);
  goodHtRequests++;
  htrvc.removeLRU(set);
}
//Case 3
else if(!mtHit && htHit && vtHit) {
  llc.setMT(set, llcLine.way);
  htrvc.setLRU(set, vtLine.way);
  htrvc.removeLRU(set);
}
//Case 4
else if(mtHit && !htHit && vtHit) {
  htrvc.setLRU(set, vtLine.way);
}
```

Figure 3. Actions Taken When Servicing an LLC Request

III. Results

Three benchmarks have been evaluated in this paper: mst and em3d in Olden, and 429.mcf in CPU2006. All three benchmarks are cross-compiled with "-O3". The HT parameters for each benchmark are set empirically to: mst: lookahead=20, blocksize=10; em3d: lookahead=10, blocksize=10; 429.mcf: lookahead=10, blocksize=10.

1) 4M LLC, 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) 4M LLC, mst 10000, HT version, checkpointed simulation
   - pending...
3) 4M LLC, em3d 1000, HT version (params: LOOKAHEAD=20), detailed simulation
   - llc.totalHtRequests=539
   - llc.goodHtRequests=517
   - llc.badHtRequests=0
   - llc.uglyHtRequests=22
4) 4M LLC, em3d 400000, HT version, checkpointed simulation
   - pending...
5) 1M LLC, mst 1000, HT version (snapshot at cycle #351552909)
   - llc.mtMisses: 1431649
   - llc.totalHtRequests: 683517
   - llc.usefulHtRequests: 683253
   - llc.ht_accuracy: 99.96%
   - llc.ht_coverage: 47.72%
   - llc.goodHtRequests: 683246
   - llc.badHtRequests: 1
   - llc.uglyHtRequests: 270
   - llc.lateHtRequests: 69075

IV. Previous Work

[3] presents a taxonomy of hardware prefetches based on the idea of shared cache pollution. A hardware structure called the Evict Table (ET) is added to the LLC to gauge the amount of shared cache pollution caused by hardware prefetching. [4]

TODOs: previous work on prefetch taxonomies.

1) Classic metrics of coverage and accuracy for h/w prefetches.
2) Good, bad and ugly breakdown of h/w prefetches.
3) More fine-grained breakdowns of h/w prefetches.
4) Any previous work on cache request breakdowns for helper threaded data prefetching?

V. 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] Vanderwiel and Lilja, "Data prefetch mechanisms," *CSURV: Computing Surveys*, vol. 32, 2000.

[2] V. Srinivasan, E. S. Davidson, and G. S. Tyson, "A prefetch taxonomy," *IEEE Trans. Computers*, vol. 53, no. 2, pp. 126–140, 2004. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/TC.2004.1261824

[3] 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

[4] N. D. E. Jerger, E. L. Hill, and M. H. Lipasti, "Friendly fire: understanding the effects of multiprocessor prefetches," in *ISPASS*. IEEE Computer Society, 2006, pp. 177–188. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/ISPASS.2006.1620802