Dynamic LLC Replacement for Helper Threaded Data Prefetching on CMPs

Min Cai, Zhimin Gu

***Abstract*—Helper threaded data prefetching on chip multiprocessors (CMPs), in its most basic form, uses the processing power of underutilized neighboring cores to improve the performance of a single-threaded program running on a shared cache CMP. Effective helper threaded data prefetching demands that the helper thread (HT) should issue correct and timely LLC requests just before the main thread (MT) requests them. However this ideal case can not be assumed in the practical implemention of helper threaded data prefetching on CMPs: some LLC requests coming from HT could not contribute to the MT performance, but instead stress and pollute LLC if no effective LLC replacement techniques are employed. Traditional notions of accuracy and coverage for hardware based data prefetching can be applied to helper threaded data prefetching on CMPs as well, however they do not provide sufficient details on the cache pollution and inter-thread LLC interference caused by HT LLC requests for improving LLC replacement for helper threaded data prefetching on CMPs.**

In this paper, we present the dynamic LLC replacement technique for helper threaded data prefetching on CMPs, based on a pollution-aware taxonomy of HT LLC requests. Firstly, the experimental methodology used in this work are discussed. Secondly, a pollution aware taxonomy of HT LLC requests is presented from the view of the contribution of HT LLC requests to the MT performance. Thirdly, based on the intuition of the result of the taxonomy applied to the Pthreads based helper threaded version of the memory-intensive mst benchmark in Olden, the dynamic LLC replacement for helper threaded data prefetching on CMPs are elaborated. Experimental results from cycle accurate simulation show that: (1) there is non-trivial LLC pollution caused by HT in helper-threaded data prefetching on CMPs; (2) LLC request taxonomy based dynamic LLC replacement can improve the effectiveness and timeliness of helper-threaded data prefetching on CMPs.

***Index Terms*—chip multiprocessors, helper threaded data prefetching, cache replacement, cache pollution, inter-thread interference**

I. Introduction

HELPER threaded data prefetching on chip multiprocessors (CMPs), in its most basic form, uses the processing power of underutilized neighboring cores to improve the performance of a single-threaded program running on a shared cache CMP. Effective helper threaded data prefetching demands that the helper thread (HT) should issue correct and timely LLC requests just before the main thread (MT) requests them. However this ideal case can not be assumed in the practical implemention of helper threaded data prefetching on CMPs: some LLC requests coming from HT could not contribute to the MT performance, but instead stress and pollute LLC if no effective LLC replacement techniques are employed.

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.

Several metrics have been proposed in the past for evaluating the effectiveness of hardware based data prefetching, among which prefetch accuracy and coverage are the most intuitive ones [1]. We can easily adapt the definitions of accuracy and coverage for hardware based data 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 [1]. 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 the cache pollution and inter-thread interference caused by HT LLC requests, which are two kinds of deficiencies in the HT scheme.

In this paper, we present the dynamic LLC replacement technique for helper threaded data prefetching on CMPs, based on a pollution-aware taxonomy of HT LLC requests. Firstly, the experimental methodology used in this work are discussed. Secondly, a pollution aware taxonomy of HT LLC requests is presented from the view of the contribution of HT LLC requests to the MT performance, in the aim of providing insights on designing and implementing effective LLC replacement techniques for the HT scheme. Thirdly, based on the intuition of the result of the taxonomy applied to the Pthreads based helper threaded version of the memory-intensive mst benchmark in Olden, the dynamic LLC replacement for helper threaded data prefetching on CMPs are elaborated. Experimental results from cycle accurate simulation show that: (1) there is non-trivial LLC pollution caused by HT in helper-threaded data prefetching on CMPs; (2) LLC request taxonomy based dynamic LLC replacement can improve the effectiveness and timeliness of helper-threaded data prefetching on CMPs. 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 answers for the following two questions:

1) How can the HT LLC requests be classified based on the

cache pollution and inter-thread LLC interference caused by HT LLC requests in helper threaded data prefetching on CMPs (or the HT scheme)?

2) How can LLC replacement be improved based on the observed result of the proposed pollution-aware HT LLC request taxonomy in the HT scheme?

The rest of this paper is organized as follows. Section 3 presents the experimental methodology used in this work. Section 2 presents the pollution aware taxonomy of HT requests and the experimental result of mst in Olden. Section 4 discusses the dynamic LLC replacement for helper threaded data prefetching on CMPs and its experimental results. Section 5 talks about related work. In section 6 we conclude the paper.

II. Experimental Methodology

We use an in-house CMP architectural simulator named Archimulator in our experiments mentioned in this work. Archimulator is a flexible execution-driven architectural simulator written in Java and running on Linux. It provides fast forward functional simulation and cycle-accurate application-only simulation of MIPSII executables on multicore architectures consisting of out-of-order super-scalar cores and configurable memory hierarchy of directory-based MESI coherence. It has basic support of simulating Pthreads based multithreaded workloads.

A. Simulated CMP Architecture

As shown in Fig.1, the simulated target CMP architecture has two cores where each core is a two-way SMT with its own private L1 caches (32KB 8-way data caches and 32KB 4-way instruction caches). Both cores share a 4MB 8-way L2 cacche. MESI coherence is maintained between L1 caches. An LRU cache called HTRVC is attached to the LLC(L2) to implement the taxonommy of HT LLC requests, which will be detailed in the next section. Detailed microarchitecture parameters are listed in Tab.I.

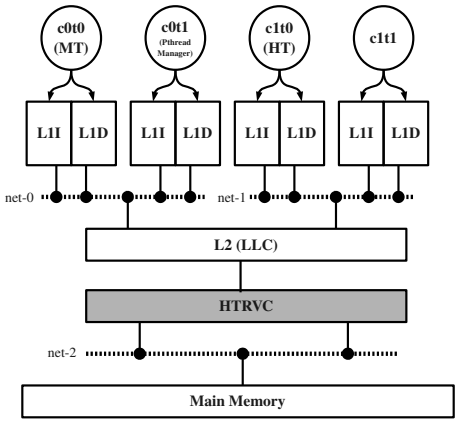

Figure 1. Simulated CMP Architecture

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 I
Baseline Hardware Configurations

B. Software Context to Hardware Thread Mapping

In 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 $\rightarrow$ MT, C0T1 $\rightarrow$ Pthreads manager thread, C1T0 $\rightarrow$ HT (C = core, T = thread), as shown in Fig.1. We use this context mapping in the following discussions.

C. ROI Based Two-Phase Fast Simulation of Helper Threaded Workloads

As shown in Fig.2, here we use two-phase simulation strategy to reduce simulaton time of memory-intensive benchmark executions which often take very long to complete. We utilize the special case when a MIPS32 assembly instruction `addiu` is invoked with register operands being R0: `addiu R0, R0, imm` to indicate the spawning point of the helper thread (named by "HT Spawn": `addiu R0, R0, 3720`). Since the code executed after encountering the "HT spawn" is our region of interest (ROI) in this work, code execution before encountering the "HT spawn" can be simulated functionally without the hassle of modeling the microarchitecture details while not hurting the experimental results. To ensure caches are warmed up, we simulate in the detailed simulation mode 100 million instructions in the main thread after the "HT Spawn" pseudocall is encountered.

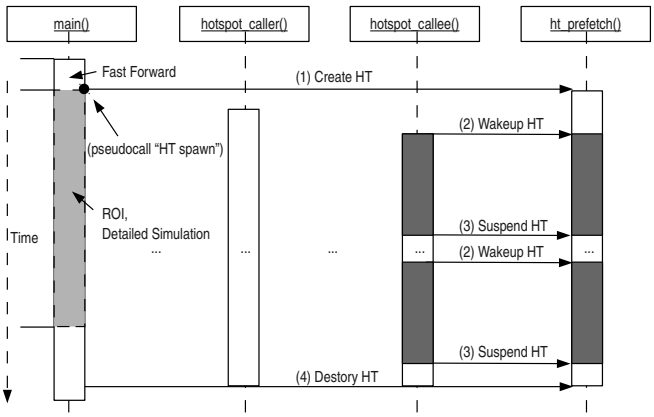

Figure 2. ROI Based Two Phase Fast Simulation of Helper Threaded Data Prefetching on CMPs

III. Pollution Aware Taxonomy of HT LLC Requests

A. A Taxonomy Based on HT-MT Inter-Thread and MT Intra-Thread Reuse Distances

We propose a pollution aware taxonomy of HT LLC requests based on the observed cache pollution caused by HT LLC requests.

The notion of inter-thread reuse distance is proposed to help explain the taxonomy. The traditional intra-thread reuse distance of a reference to data element x is defined as the number of distinct data elements that have been referenced between two contiguous accesses of x (or ∞ if the element has not been referenced thereafter). These two contiguous accesses are issued by the same thread in the traditional notion. To accomodate the case where one thread Tb accesses data element x that has been previously brought into the cache by another thread Ta, we introduce the notion of Ta-Tb inter-thread reuse distance, as compared to the traditional intra-thread reuse distance. Therefore, we could use HT-MT inter-thread reuse distance to describe the case where the main thread accesses data element x that has been previously brought into the LLC by the helper thread.

Based on the notions of HT-MT inter-thread reuse distance (HT-MT_ITRD) and MT intra-thread reuse distance (MT_RD), we can classify the HT LLC requests into three types: good, bad and ugly. Let's assume that when an HT LLC request with data d evicts the LLC line containing victim data v which is brought by MT, and afterwards the LLC line containing d is evicted by an MT LLC request with data m, then we can see that:

1) HT-MT_ITRD(d) < MT_RD(v) < MT_RD(m) , indicating it's a good HT request;
2) MT_RD(v) < HT-MT_ITRD(d) < MT_RD(m) , indicating it's a bad HT request;
3) MT_RD(m) < HT_MT_ITRD(d), MT_RD(m), indicating it's an ugly HT request.

As illustrated in Tab.II, among these three 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.

Among good HT requests, late HT requests should be differentiated, which are in-flight HT LLC requests that are hit by MT LLC requests, as they are good indicators for potential performance improvement of the HT scheme because late HT requests may be converted to timely HT requests by finetuning the parameters of the HT scheme.

Table II
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

B. Hardware Support for the Pollution-Aware Taxonomy of HT LLC Requests

1) 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 [2].

2) 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).

3) 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 [2], 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.

4) 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 equivalent hardware components.

C. Algorithms for Tracking HT LLC 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;

a) # 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:

a) An HT request evicts an INVALID line. In this case, no eviction is needed.
b) 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.
c) 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.
d) 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.
e) 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.3, 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:

a) LLC miss and victim hit, which indicates a bad HT request. This happens when HT request evicts useful data.
b) HT LLC hit, which indicates a good HT request. This happens when HT requested data is hit by MT request before evicted data.
c) HT LLC hit and victim hit. This happens when useful data is evicted and brought back in by HT request.
d) MT LLC hit 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.4, 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.

```
//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 3. Actions Taken When Fillling an LLC Line

```
//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 4. Actions Taken When Servicing an LLC Request

D. Results

We use the original version and helper threaded version of mst as workload used in our experiments. All benchmarks are cross-compiled with gcc flag "-O3".

Tab.III summarizes overall performance of the benchmark in which all the speedups are computed against the baseline original version of the mst; K = lookahead, P = blocksize.

Table III
MST HT LLC REQUEST TAXONOMY RESULTS

K, P	cycles	speedup	MT LLC Read Misses	HT LLC Requests			
				Good	Bad	Ugly	Late
L2: 1M, 4-way							
original	2128177521						
10, 10	1413640215			5314478	9	15831	2291709
20, 10	1201000213			4596361	25	33536	571166
20, 20	1391083752			5132796	0	514	2007020
320, 320	1299136860			4629522	449	95403	1049485
640, 320	1281837899			3658166	1252	720821	190
640, 640	1342389544			4021349	1697	1053477	633125
L2: 1M, 8-way							
original	2128379206						
10, 10	1413844837			5300159	35	28713	2278392
20, 10	1201909631			4602015	28	28613	580520
20, 20	1394465307			5110279	86	32673	1996862
320, 320	1298792424			4642918	577	62818	1059197
640, 320	1281744764			3661219	1939	703163	226
640, 640	1337983575			4043866	2526	1017277	633273
L2: 2M, 4-way							
original	2120356167						
10, 10	1399634774			5257629	26	30005	2305058
20, 10	1174856777			4516140	0	319	488142
20, 20	1373584291			5031697	16	11407	1943799
320, 320	1309023530			4724771	83	9234	1302455
640, 320	1310880244			3462836	595	670785	35252
640, 640	1336338360			4244832	714	681175	932923
L2: 2M, 8-way							
original	2120668282						
10, 10	1394816682			5280283	0	295	2335138
20, 10	1167492039			4485981	0	522	446726
20, 20	1374797661			5039645	6	8408	1972291
320, 320	1321725531			4753262	130	597	1410322
640, 320	1293183359			3862339	302	77499	388081
640, 640	1315342952			4673206	354	75351	1302622
L2: 4M, 4-way							
original	2114816878						
10, 10	1389796215			5250168	0	131	2323420
20, 10	1142128970			4396365	0	167	274300
20, 20	1368304714			5003078	0	178	1945235
320, 320	1325248401			4720657	34	679	1434343
640, 320	1319660419			3510111	403	499676	183632
640, 640	1335035605			4334291	356	504965	1078711
L2: 4M, 8-way							
original	2112469520						
10, 10	1387072573			5235217	4	5389	2320005
20, 10	1159240286			4451922	0	284	435001
20, 20	1369694579			4991352	2	6000	1958224
320, 320	1330619790			4720528	21	458	1478493
640, 320	1293317577			3884985	161	11982	464030
640, 640	1319089645			4692382	195	10284	1393115

IV. DYANAMIC LLC REPLACEMENT FOR HELPER THREADED DATA PREFETCHING ON CMPS

A. Algorithms

B. Results

V. RELATED WORK

[2] presents a taxonomy of hardware prefetches based on the idea of shared cache pollution in the hardware based data prefetching for shared L2 CMP. A hardware structure called the Evict Table (ET) is attached to the LLC to gauge the amount of shared cache pollution caused by hardware prefetching. [3]

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?

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] 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

[2] 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

[3] 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