Feedback Directed Helper Threaded Inter-Core Data Prefetching

Min Cai, Zhimin Gu

***Abstract*—For applications exhibiting irregular memory access patterns, helper threaded inter-core data prefetching on shared cache chip multiprocessors (CMPs) speculatively issue LLC requests to the predicted memory addresses before the computation core access these addresses. Effective helper threaded inter-core data prefetching demands that the helper thread (HT) should issue correct and timely LLC requests just before the main thread (MT) requests them. Unfortunately, this ideal case can not be assumed in the existing implementation of helper threaded inter-core data prefetching: inaccurate and/or untimely LLC requests coming from HT could not contribute to the MT performance, but instead stress and pollute LLC if no effective LLC replacement and pollution-aware feedback techniques are employed.**

In this paper, we present a pollution-aware feedback mechnism for dynamic helper threaded inter-core data prefetching, based on a pollution-aware taxonomy of HT LLC requests. A pollution aware taxonomy of HT LLC requests is presented from the view of the contribution of HT LLC requests to the MT performance. Based on the pollution aware taxonomy of HT LLC requests, the feedback mechanism for dynamic helper threaded inter-core data prefetching is proposed to finetune LLC replacement and parameters of the HT scheme. Experimental results from cycle accurate simulation of the Pthreads based helper threaded version of the memory-intensive mst benchmark in Olden 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 inter-core data prefetching, cache replacement, cache pollution, inter-thread interference**

I. Introduction

FOR applications exhibiting irregular memory access patterns, helper threaded inter-core data prefetching on shared cache chip multiprocessors (CMPs), in its most basic form, (1) uses the shared cache enabled fast inter-thread communication when both the main core and the otherwise idle neighboring helper core access the same cache line; (2) constructs a helper thread in the software running on the helper core to speculatively issue LLC prefetch requests to the predicted memory addresses before the computation core access these addresses. 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. Unfortunately, this ideal case can not be assumed in the existing implementation of helper threaded inter-core data prefetching: inaccurate and/or untimely 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.

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 traditional accuracy and coverage metrics for HT requests and the classification of "useful" and "useless" HT requests don't care about the LLC 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 a pollution-aware feedback mechnism for dynamic helper threaded inter-core data prefetching, based on a pollution-aware taxonomy of HT LLC requests. A pollution aware taxonomy of HT LLC requests is presented from the view of the contribution of HT LLC requests to the MT performance. Based on the pollution aware taxonomy of HT LLC requests, the feedback mechanism for dynamic helper threaded inter-core data prefetching is proposed to finetune LLC replacement and parameters of the HT scheme. Experimental results from cycle accurate simulation of the Pthreads based helper threaded version of the memory-intensive mst benchmark in Olden 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

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.

cache pollution and inter-thread LLC interference caused by HT LLC requests in helper threaded inter-core data prefetching (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 2 presents the pollution aware taxonomy of HT requests and the experimental result of mst in Olden. Section 3 discusses the dynamic LLC replacement for helper threaded inter-core data prefetching and its experimental results. Section 4 presents the experimental methodology used in this work. Section 5 discusses results. Section 6 talks about related work. In section 7 we conclude the paper.

II. Background and Motivation

A. The Scheme of Helper Threaded Inter-Core Data Prefetching on Shared Cache CMPs

As illustrated in Fig.1, the workflow of helper threaded inter-core data prefetching we implement here can be described as follows.

1) the helper thread is spawned in the entry point `main()` of the program;
2) the helper thread remains dormant until some caller of the target hotspot function has been invoked and code placed in the caller wakes up the helper thread to let it start the `prelude` where the code in the helper thread skips some iterations of pointer traversals (i.e., there is no prefetch issued) to compensate the long time used for data prefetching in the helper thread as compared to the short time used in computation work in the main thread;
3) the helper thread enters a `steady state` of issuing LLC prefetch requests in loop iterations of pointer traversals ahead of the main thread until the execution of the program has passed some point(s) in the target hotspot function;
4) the code in helper thread is synchronizing pointers with the main thread and begin the next turn of servicing hotspots;
5) after all the prefetching work is done, the helper thread is destroyed in main().

Two parameters in the HT scheme controls its aggressiveness: (1) the number of loop iterations of pointer traversals that the helper thread code skip after synchronizing with the main thread in the prelude is called `lookahead`; (2) the number of loop iterations of pointer traversals in which helper thread code issue LLC prefetch requests in the steady state is called `stride`.

In a traditional HT scheme configuration, the values of `lookahead` and `stride` are hard coded. In our following proposed feedback directed mechanism, the processor changes the value of `lookahead` and `stride` using some temporary register to adjust the aggressiveness of the HT scheme.

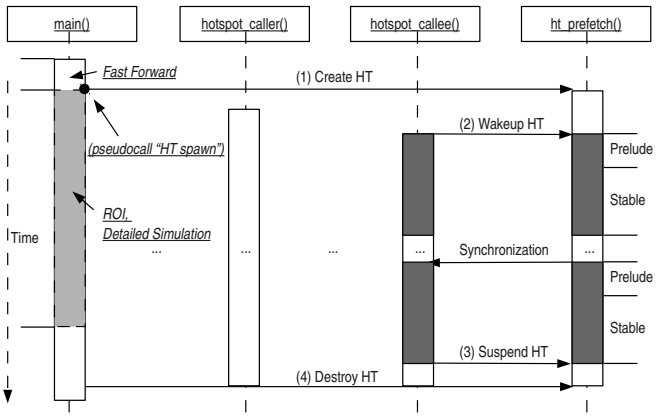

Figure 1. The Scheme of Helper Threaded Inter-Core Data Prefetching

B. Pollution-Unaware Metrics For Evaluating the Effectiveness of the HT Scheme

Traditionally, prefetch accuracy, coverage and lateness are used to evaluate the effectiveness of hardware prefetchers. In this section, we adapt the metrics to the HT scheme and describe what the metrics mean in the effectiveness of the HT scheme.

1) HT LLC Request Accuracy: HT LLC request accuracy is a measure of how accurately the HT scheme can predict and issue prefetch requests for the memory addresses that will be accessed by the main thread. It is defined as below

$$HT\ LLC\ Request\ Accuracy = \frac{Number\ of\ Useful\ HT\ LLC\ Requests}{Number\ of\ HT\ LLC\ Requests}$$

where Number of Useful HT LLC Requests is the number of LLC lines brought by HT requests that are later on hit by MT requests. For benchmarks with high HT LLC request accuracy, performance increases as the aggressiveness of the HT scheme is increased.

2) HT LLC Request Coverage: HT LLC request coverage is a measure of the fraction of all MT LLC misses that can be eliminated by issuing HT LLC misses ahead. It is defined as below

$$HTLLCRequestCoverage = \frac{Number\ of\ Useful\ HT\ LLC\ Requests}{Number\ of\ MT\ LLC\ Requests\ in\ No\ HT}$$

3) HT LLC Request Lateness: HT LLC request lateness is a measure of how timely the LLC requests generated in the helper thread are with respect to the LLC requests generated in the main thread that need the data brought by the helper thread. An HT LLC request is said to be late if its requested data has not yet returned from the main memory by the time an MT LLC request references the data. Therefore, even though the HT LLC request is accurate, it can only partially hide the latency incurred by an LLC miss in the main thread. *HT LLC request lateness* can be defined as below

$$HT\ LLC\ Request\ Lateness = \frac{Number\ of\ Late\ HT\ LLC\ Requests}{Number\ of\ Useful\ HT\ LLC\ Requests}$$

III. Pollution Aware Taxonomy of HT LLC Requests

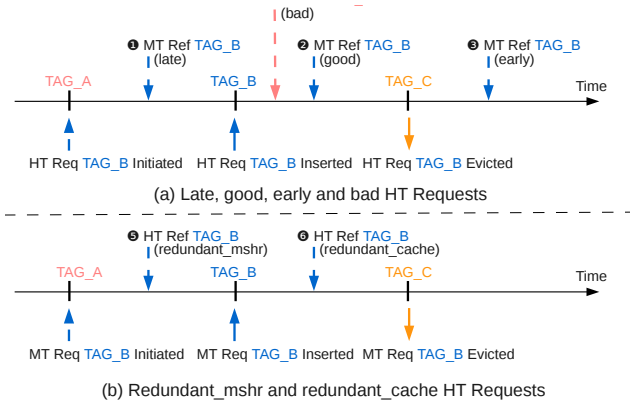

Figure 2. Taxonomy of HT LLC Requests

A. The Notion of Inter-Thread Reuse Distance in the HT Scheme

The notion of reuse distance can shed light on how a pollution-aware taxonomy of HT LLC requests can be constructed. The traditional intra-thread reuse distance of a reference to data element x is defined as the number of unique memory references between two consecutive accesses of x in the same thread (or ∞ if the element has not been referenced thereafter). In a k-way set-associative cache a cache miss with reuse distance rd can be classified by the value of reuse distance as below

1) rd < k indicates it is a conflict miss
2) k<= rd < ∞ indicates it is a capacity miss
3) rd = ∞ indicates it is a cold miss

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 applied to either Ta or Tb.

To show why the notion of inter-thread reuse distance is important in the HT scheme as compared to the traditional notion of intra-thread reuse distance, we can consider the mst benchmark in the Olden suite. Its pointer traversing code structure inherently exhibits irregular memory access pattern in its original, single threaded version, which renders the common LRU replacement policy inefficient to reduce LLC misses. Here we use RD_{MT} to refer to the *intra-thread reuse distance* in the main thread, and $ITRD_{HT-MT}$ to refer to the *inter-thread reuse distance* between HT and MT where a data element is first brought to LLC by HT, and later on used by MT. Therefore, the values of RD_{MT} in most MT LLC requests is high which reflects the irregular memory access pattern in mst. The values of $ITRD_{HT-MT}$ in HT LLC requests should be very small when the HT scheme is efficient to reduce LLC misses in MT where for most HT LLC miss, an immediate followup MT LLC request will access the data brought by the previous LLC request and hit in the LLC. Otherwise, large values of $ITRD_{HT-MT}$ indicate the inefficiency of the HT scheme where most data brought by HT is replaced before used by MT.

HT induced cache pollution with respect to MT performance only happens when HT LLC requests evict the data that are previously brought by MT and immediately referenced again by MT LLC requests, but rarely happens when HT LLC requests evict any data that was previously brought by MT but will not be used by MT in the near future, is evicted by HT, which is typically the case in our selected benchmark `mst` which exhibits a thrashing memory access pattern and thus most of the data requested from its delinquent PCs have instant MT intra-thread reuse distances but small HT-MT inter-thread reuse distances, which renders traditional intra-thread reuse distance prediction based LLC replacement useless for `mst` with HT. Fortunately, as we will see, HT-MT inter-thread reuse distance prediction based LLC replacement can be useful for `mst` with HT.

B. Pollution Aware Taxonomy of HT LLC Requests

Based on the notions of HT-MT inter-thread reuse distance ($ITRD_{HT-MT}$) and MT intra-thread reuse distance (RD_{MT}), we can construct a pollution aware taxonomy of HT LLC requests where HT LLC requests are classified into three types: good, bad and ugly. Let's assume when an HT LLC request referencing data `h` evicts an LLC line containing the victim data `v` which is brought by MT, and afterwards the LLC line containing `h` is evicted by an MT LLC request with data `m`. Therefore, there are two LLC replacement involved: first `h` evicts `v` and then `m` evicts `h`. We use $RD_{MT}(v)$, $ITRD_{HT-MT}(h)$ and $RD_{MT}(m)$ to denote the number of distinct data elements that have been referenced between the time when `h` evicts `v` and the time `v`, `h` and `m` is accessed again, respectively. The greater the reuse distance of the data, the farther the data will be referenced again in time. An LLC replacement is considered as optimal if the replacement makes the data with larger reuse distance evicts the data with smaller reuse distance, otherwise the replacement is considered as un-optimal. We have

1) if $ITRD_{HT-MT}(h) < RD_{MT}(v) < RD_{MT}(m)$, then the HT LLC request is considered as *good* because it evicts the data that has larger reuse distance than its data. Its data is hit by MT before evicted. It has posive impact on MT performance since it reduce one MT miss;
2) if $RD_{MT}(v) < ITRD_{HT-MT}(h) < RD_{MT}(m)$, then the HT LLC request is considered as *bad* because it evicts the data that has smaller reuse distance than its data. It displaces an LLC line that will later be needed by MT. It is harmful for MT performance which should be prevented as much as possible;
3) if $RD_{MT}(m) < RD_{MT}(v)$ and $ITRD_{HT-MT}(h)$, then the HT LLC request is considered *ugly* because both its data and its victim data have larger reuse distances than the data h. It has little performance impact on MT performance because the requested data is not referenced by MT before evicted and it does not evict any data that will be used by MT.

We can see that, good and bad HT LLC requests are caused by the optimal and un-optimal LLC replacement, respectively. We can conclude that

1) Low $ITRD_{HT-MT}$ is an indicator of good performance of the HT scheme;
2) Medium$ITRD_{HT-MT}$ signals those potentially late HT prefetches that, depend on the LLC replacement policy, may be useful, partially useful or useless for MT performance;
3) High $ITRD_{HT-MT}$ implies HT and MT is not synchronized well or the amount of (computational and memory access) work in HT and MT is not balanced well.

As noted in the previous section, good HT LLC requests can be further divided into timely requests and late requests. An HT request is called `late HT request` if its requested data has not yet returned from the main memory by the time an MT LLC request references the data. Late HT requests only partially hide the LLC miss latency in MT requests, but they are good indicators for potential performance improvement of the HT scheme because late HT requests may be converted to timely HT requests by adjusting the aggressiveness parameters of the HT scheme.

C. 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 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 Miss Status Holding Register (MSHR). MSHR is a hardware structure that keeps track of all in-flight memory requests. An HT LLC request is late if an MT LLC request for the same address is generated while the HT LLC request is in the LLC MSHR waiting for main memory.

D. 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;
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.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:

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

IV. Pollution-Aware Feedback Directed Helper Threaded Inter-Core Data Prefetching

A. Inter-Thread Reuse Distance Based HT Quality Assessment

Under the LRU LLC replacement policy, for LLC misses that issued from delinquent PCs in either in MT or HT, the value of HT-MT inter-thread reuse distance is (w.rt. the associativity of the cache)

- HT-MT ITRD >= 0
  - ITSD > assoc ⇒ replaced HT prefetches (need aggressiveness tuning)
  - ITSD <= assoc
    - Cache hit ⇒ useful HT prefetches
    - MSHR hit ⇒ late$_1$ HT prefetches (need aggressiveness tuning)
- HT-MT ITRD <0
  - Low ⇒ late$_1$ HT prefetches (need aggressiveness tuning)
  - Medium to High ⇒ late$_2$ HT prefetches (need synchronization)
  - Distant ⇒ MT loads not covered by HT prefetches

B. Inter-Thread Reuse Distance (ITRD) Prediction

- For each program phase separated by two consecutive synchronizations, in the end of the program phase
  - ITRD = 1/2 * previous_ITRD + 1/2 * ITRD_in_the_current_phase
- A confidence counter with a threshold value of 10 and maximum value of 20
- Main components, similar to the one used in the reuse distance prediction based cache replacement paper
  - Data address indexed FIFO backed sampler
  - PC indexed LRU cache backed predictor (has the same geometry as L2)

C. Inter-Thread Reuse Distance Based LLC Replacement for the HT scheme

- Extra hardware: an additional HT bit per L2 line
  - Each cache line has an additional HT bit indicating the line is brought by HT and has not been used by MT yet
  - When a cache line marked by HT is referenced by MT, it is unmarked
- Insertion position of an L2 read miss issued from delinquent PCs and marked as HT, if its predicted HT-MT reuse distance is
  1) high ⇒ just bypass the L2 or LRU position
  2) medium ⇒ middle position or 1/4 LRU position
  3) low ⇒ MRU position
  4) cannot be determined ⇒ LRU position
- Insertion position of an L2 read miss issued from delinquent PCs and not marked as HT: LRU position
- Insertion position of an L2 read miss in remaining cases: MRU position

D. Inter-Thread Reuse Distance Based feedback directed synchronization triggering and aggressiveness adjustment for the HT scheme

- Hardware triggered synchronization of HT with MT
  - Based on the already implemented pseudo call mechanism that pass parameters between the program and the processor in some unused registers by the ADDIU MIPS instruction with predefined special parameters
  - A register based polling process is used to let the HT periodically ask the processor if it need to synchronize with MT
  - Tackling the B parameter in K-P-B model

  - Can potentially utilize the synchronization register mecahnisms proposed in existing papers to reduce synchronization overhead
- Dynamic HT aggressivenss adjustment
  - Parameters for adjusting the aggressiveness of the HT are
    * Lookahead (a.k.a. the K parameter in K-P-B model)
    * Prefetching work within one program phase (a.k.a. the P parameter in K-P-B model)

V. Evaluation 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.5, 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 cache. MESI coherence is maintained between L1 caches. An LRU cache called HTRVC is attached to the LLC(L2) to implement the taxonomy of HT LLC requests, which will be detailed in the next section. Detailed microarchitecture parameters are listed in Tab.I.

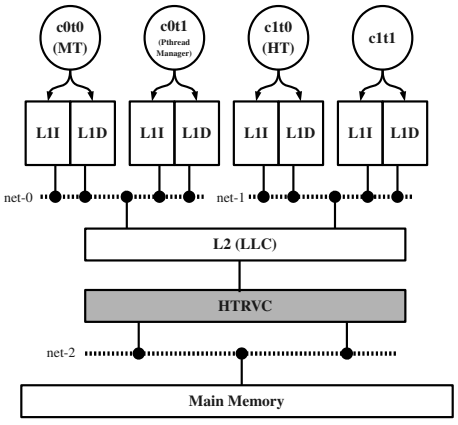

Figure 5. Simulated CMP Architecture

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, suspending 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.5. We use this context mapping in the following discussions.

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

As shown in Fig.1, 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" pseudo call is encountered.

VI. Results

We use the original version and helper threaded version of mst as workload used in our experiments. All versions of mst

benchmark are cross-compiled with gcc flag "-O3". Input args are "4000".

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

Table II
MST HT LLC REQUEST TAXONOMY RESULTS UNDER LRU LLC REPLACEMENT

K, P	cycles	speedup	MT LLC Read Misses	HT LLC Requests			
				Good	Bad	Ugly	Late
L2: 1M, 4-way							
Unmodified	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							
Unmodified	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							
Unmodified	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							
Unmodified	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							
Unmodified	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							
Unmodified	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

VII. RELATED WORK

Even though helper threaded data prefetching mechanisms have been studied for a long time, taxonomy of HT LLC requests have not been studied before. Here we briefly describe previous work in the context of hardwre prefething.

Table III
MST HT LLC REQUEST TAXONOMY RESULTS UNDER DYNAMIC LLC REPLACEMENT

K, P	cycles	speedup	MT LLC Read Misses	HT LLC Requests			
				Good	Bad	Ugly	Late
L2: 1M, 4-way							
Unmodified							
10, 10	1415822623			5289283	24	49899	2277062
20, 10	1198389681			4602585	14	17316	564991
20, 20	1391086968			5133595	0	516	2007792
320, 320	1297462890			4643525	455	78974	1055340
640, 320	1281823292			3658069	1279	721139	169
640, 640	1342388863			4020819	1708	1053730	632577
L2: 1M, 8-way							
Unmodified							
10, 10	1414936009			5287475	46	45585	2270900
20, 10	1201422011			4610353	23	17889	586688
20, 20	1394761071			5104755	85	36819	1992954
320, 320	1298818263			4642534	560	65117	1059315
640, 320	1281878608			3659494	1953	708409	249
640, 640	1339670226			4037280	2490	1030596	634789
L2: 2M, 4-way							
Unmodified							
10, 10	1399638510			5257106	25	30014	2304490
20, 10	1177357943			4522885	0	317	507372
20, 20	1373593890			5033452	24	11405	1945536
320, 320	1309033612			4724233	81	9391	1301979
640, 320	1310831837			3462866	586	671131	35181
640, 640	1336354341			4243643	683	681561	931757
L2: 2M, 8-way							
Unmodified							
10, 10	1394822639			5280466	0	292	2335433
20, 10	1168890592			4489385	0	538	456889
20, 20	1374802543			5039894	6	8407	1972496
320, 320	1321707262			4753309	128	622	1410366
640, 320	1293146652			3862072	288	77652	387710
640, 640	1315353076			4674418	378	75388	1303841
L2: 4M, 4-way							
Unmodified							
10, 10	1389796412			5250450	0	132	2323455
20, 10	1140630504			4391249	0	163	261977
20, 20	1368310342			5001750	0	177	1943891
320, 320	1325261372			4720049	33	804	1433818
640, 320	1319659301			3509582	412	499934	183136
640, 640	1335042038			4337037	361	505222	1081397
L2: 4M, 8-way							
Unmodified							
10, 10	1387074688			5234229	4	5388	2319043
20, 10	1158143928			4447738	0	276	425326
20, 20	1369698439			4991788	3	6000	1958674
320, 320	1330618620			4720103	21	458	1478065
640, 320	1293325892			3886050	162	12020	465096
640, 640	1319104804			4692545	196	10290	1393320

A. Reuse Distance

B. Prefetch Taxonomies

Our taxonomy of HT LLC requests is mostly similar to the work in [2], which 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. Good, bad and ugly requests are identified based on cache replacement activities involved by hardware prefetches. [3] developed a multiprocessor prefetch traffic and miss taxonomy that builds on existing uniprocessor taxonomy.

C. Prefetch Aware Cache Content Management

[4] characterizes the performance of state-of-the-art LLC management policies in the presence and absence of hardware prefetching. Prefetch-Aware Cache Management (PACMan) is proposed to dynamically estimates and mitigates the degree of prefetch-induced cache interference by modifying the cache insertion and hit promotion policies to treat domand and prefetch requests differently.

D. Feedback Directed Data Prefetching

Our feedback-directed scheme is similar to the work in [5], which proposes a low-cost feedback directed mechanism for hardware prefetching. The mechanism can be applied to any hardware prefetchers such as sequential prefetchers, stream-based prefetchers, GHB based prefetchers and PC-based stride prefetchers.

VIII. Conclusion

This paper proposed the pollution-aware taxonomy of LLC requests issued by the helper thread (HT) in the helper threaded inter-core data prefetching, and discussed a feedback directed runtime mechanism based on the taxonomy that dynamically adjusts the LLC replacement policy and the aggressiveness of a Pthreads based HT implementation to reduce cache pollution caused by HT LLC requests and increase the timeliness of HT LLC requests, thus improve the effectiveness of the HT scheme.

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.

[2] B. Mehta, D. Vantrease, and L. Yen, "Cache showdown: The good, bad and ugly," Tech. Rep., 2004.

[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.

[4] C.-J. Wu, A. Jaleel, M. Martonosi, S. C. S. J. , and J. S. Emer, "Pacman: prefetch-aware cache management for high performance caching," in *MICRO*, ser. 44rd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2011, 3-7 December 2011, Porto Alegre, Brazil, C. Galuzzi, L. Carro, A. Moshovos, and M. Prvulovic, Eds. ACM, 2011, pp. 442–453. [Online]. Available: http://doi.acm.org/10.1145/2155620.2155672

[5] S. Srinath, O. Mutlu, H. Kim, and Y. N. Patt, "Feedback directed prefetching: Improving the performance and bandwidth-efficiency of hardware prefetchers," in *Proc. 13th International Conference on High-Performance Computer Architecture (13th HPCA'07)*. San Francisco, CA, USA: IEEE Computer Society, feb 2007, pp. 63–74.