Performance Modeling for Helper Thread on Shared Cache CMPs

Min Cai, Zhimin Gu*, Yinxia Fu
School of Computer Science and Technology
Beijing Institute of Technology
Beijing, PR China
* Corresponding author Email : zmgu@x263.net

***Abstract*---In data intensive applications of Cloud Computing such as XML parsing, large graph traversing and so on, there are a lot of operations to access irregular data. These data need be timely prefetched into the shared cache in CMPs by helper thread. However, a bad prefetching strategy of helper thread will cause the multi-core shared cache pollution and degradation of performance. For analyzing the stand or fall of prefetching strategy of helper thread, this paper proposes a performance model for helper thread prefetching, then aiming at relevant to standard testing programs in Olden and CPU2006, the experimental results show that our model is effective for identifying the distribution of timely, late, bad, ugly of helper thread LLC requests. We could find the stand or fall of prefetching strategy of helper thread by observing the foremost factors in the distribution.**

Keywords--multi-core shared cache; Helper thread; Performance model

I. INTRODUCTION

Data intensive workloads of Cloud computing such as XML parsing and irregular graph traversing exhibit irregular memory access patterns which makes traditional prefetching techniques useless. For improving their performance, the proliferation of shared cache multiprocessors (CMPs) [4] enables helper threads [5] running on the idle cores of CMPs speculatively issue last-level cache (LLC) prefetches [1] to the predicted memory addresses, in order to hide the latencies of main thread references. Effective helper thread prefetching (HT) on CMPs demands that the helper thread should issue accurate and timely LLC requests early enough before the main thread references them. Unfortunately, the real helper thread prefetching strategies are not always effective, inaccurate and/or untimely LLC requests coming from the helper thread could not contribute to the main thread performance but rather stress and pollute the LLC. This phenomenon is witnessed by the previous studies on cache management for hardware prefetching [7].

The basic workflow of helper thread prefetching on shared-cache CMPs can refer to [3, 5]. In general, there are three helper thread prefetching parameters to control the helper thread's aggressiveness and effectiveness: Lookahead, Stride and Synchronization Period. However, the traditional helper thread prefetching approach selects empirically the prefetching parameter values. It is unable to accommodate the unpredictable nature of the LLC runtime behavior in the presence of helper thread requests. And studying the interplay of helper thread parameters and its impact on LLC activities are impossible on current real machines because of measurement costs and limited capabilities of hardware performance monitoring counters. In this paper, we tailor the open source CMP architectural simulator Archimulator to meet the needs of simulation and characterization of helper thread prefetching on CMPs, and use cycle-accurate architectural simulation to elaborate on the LLC interference caused by helper thread prefetching and its impact on the overall performance. Experimental results of data intensive, pointer traversal benchmarks from Olden [9] and CPU2006 [6] show that the appropriate value-selection of helper thread parameters such as lookahead, stride and synchronization period, plays a key role in performance improving of helper thread prefetching.

The main contributions of this paper can be summarized as follows: We propose a helper thread performance model based on reuse distance, and we could find how many helper thread LLC requests are useless or useful by this method. The induced LLC interference caused by the helper thread LLC prefetching requests have not been studied before. Our model is different from the metrics of prefetch accuracy, coverage and lateness [10], and is mostly similar to the work described in [8], which presented a taxonomy of hardware prefetches from the viewpoint of hardware prefetch. Good, bad and ugly requests are identified based on cache replacement activities involved by hardware prefetches. A hardware structure called the *Evict Table* (ET) was proposed to attach to the LLC, in order to gauge the amount of hardware prefetch induced shared cache pollution. The HTRVC structure proposed in our work is used for tracking software based helper thread L2 request victims instead of hardware prefetch victims.We show also how to tune the helper thread parameters for improving effectiveness of helper thread prefetching.

II. PERFORMANCE MODELING OF HELPER THREAD PREFETCHING

The simulated baseline CMP machine is consisted of two 2-way out-of-order SMT cores. Each core has its own private L1 caches, which are writeback and maintain inclusion with a 1MB 8-way shared L2 cache. The system implements sequential consistency using directory based MESI coherence on a basic point-to-point on-chip network. Both L1 and L2 caches use the LRU replacement policy. We developed the open source Archimulator [2] CMP architectural simulation environment for the experiments mentioned in this work. Archimulator is an object-oriented execution-driven application-only architectural simulation environment written entirely in Java, and running on Unix/Linux based operating systems. It can

simulate MIPS II executables in three modes: functional simulation, cycle-accurate simulation and two-phase fast forward and measurement simulation. It supports simulating Pthreads based multi-threaded workloads. In a typical Pthreads based helper thread program, there are three threads while running: main thread, helper thread and Pthreads manager thread. The Pthreads manager thread takes the role of spawning, suspending and resuming the helper thread by passing signals to the helper thread. The three software contexts are mapped to the hardware threads as follows: C0T0 →main thread, C0T1 →Pthreads manager thread, C1T0 →helper thread (C = core, T = thread).

A. Basic Metrics for Helper Thread Prefetching

•Useful vs. Useless Helper Thread LLC Requests. A helper thread LLC request can be either useful or useless based on the re-references of its requested data. Specifically, a helper thread request is useful if a subsequent main thread request reuses its data before the LLC displaces its data. Otherwise, a helper thread request is useless if the LLC displaces its data before a subsequent main thread request reuses its data.

•Helper Thread LLC Request Accuracy. It is of vital importance that the helper thread can accurately predict and prefetch the data that the subsequent main thread requests will use. For benchmarks with high helper thread LLC request accuracy, the overall performance increases as the aggressiveness of the helper thread prefetching scheme increases. Otherwise, for benchmarks exhibiting low helper thread LLC request accuracy, the overall performance degrades due to shared cache pollution as the aggressiveness of helper thread prefetching scheme is increased. The helper thread request accuracy is defined as # Useful HT Requests / # Total HT Requests, where the # Useful HT Requests is the number of the LLC lines brought by the helper thread requests that are re-referenced by subsequent main thread requests.

•Helper Thread LLC Request Coverage. Determining how many main thread LLC misses can be reduced by the helper thread, is insightful in fine-tuning the helper thread prefetching parameters. Specifically, Helper thread LLC request coverage is such a measure of the fraction of all main thread LLC misses in the baseline version (where helper thread is switched off) that can be converted into hits in the helper thread version. The helper thread achieves this reduction of main thread LLC misses by issuing requests early enough to convert subsequent main thread LLC misses into hits. It is defined as # Useful HT Requests / # MT Misses w/o HT.

•Helper Thread LLC Request Lateness. A helper thread LLC request is late if it fails to bring the data from the main memory by the time a main thread LLC request references the data. Therefore, even though the helper thread LLC request is accurate, it may only partially hide the latency incurred by an LLC miss in the main thread. The helper thread LLC request lateness can thus be defined as # Late HT Requests / # Total HT Requests.

•Helper Thread LLC Request Pollution. A helper thread LLC request is polluting if it evicts prematurely the useful data that will be referenced by a main thread LLC request. The helper thread LLC request pollution can thus be defined as # Polluting HT Requests / # Total HT Requests.

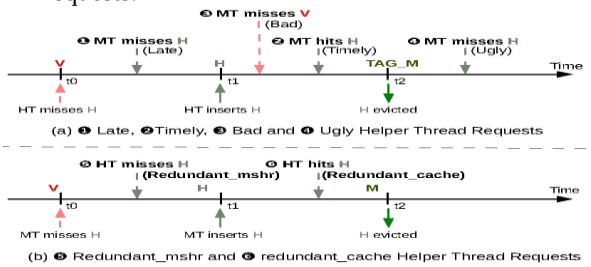

Figure 1.Timeline for the Breakdown of Helper Thread LLC Requests

B. Performance Modeling of Helper Thread Prefetching

We can construct a fine-grain breakdown of helper thread LLC requests, to express helper thread prefetch usefulness, lateness, pollution and redundancy. In Figure 1(a), there are four cases based on the relative positions of the helper thread request and the subsequent main thread request:

•**A late helper thread request**, happens when the main thread request to the data element H arrives after the helper thread request to H is initiated (t0) but before the data element H is inserted into the LLC (t1). A late helper thread request is similar to the following timely helper thread request but it only partially hides the LLC miss latency in the main thread request.

•**A timely helper thread request**, happens when the main thread request to the data element H arrives after the helper thread request's data element H is inserted into the LLC (t1) but before the data element H is evicted from the LLC (t2). This case is the result of the optimal decision made at the time the helper thread request is initiated, since it evicts the data element (V) that has larger reuse distance than its own (H). This case has the most positive impact on the main thread performance since it reduces one main thread miss.

•**A bad helper thread request**, happens when the main thread request to the victim data element V arrives after the helper thread's data element H is inserted into the LLC (t1) but before the data element H is evicted from the LLC (t2). This case is the result of the non-optimal decision made at the time the helper thread is initiated, since it evicts the data element (V) that has smaller reuse distance than its own (H). In other words, it displaces an LLC line that will later be needed by the main thread. This case should be prevented as much as possible, which is harmful for the main thread performance.

•**An ugly helper thread request**, happens when the main thread request to the data element H arrives after the helper thread request's data element H is evicted from the LLC (t2). This case is the result of the ugly decision made at the time when the helper thread request is initiated, since both its data element (H) and the victim data element (V) have larger reuse distances than the subsequent main thread data element (M). This case has little performance impact on the main thread performance other than increases the on-chip interconnect bandwidth consumption, since the helper thread's data element is not reused by the main thread

before evicted and it does not evict any data element that will be used by the main thread.

If the helper thread request is too late that it even can not catch up with the main thread, then when it arrives at the LLC, it may find the data is already present in either LLC or LLC MSHRs. Therefore, as depicted in Figure 1(b), there are two more cases based on the relative positions of the main thread request and the late helper thread request:

•**A redundant_mshr helper thread request**, happens when the helper thread rquest arrives after the main thread request is initiated (t0) but before the data element is inserted into the LLC (t1).

•**A redundant_cache helper thread request**, happens when the helper thread request arrives after the main thread request's data element is inserted into the LLC (t1) but before the data element is evicted from the LLC (t2). These two cases contribute nothing but the waste of the on-chip interconnect bandwidth to the overall performance.

Hence we have the following theorem1.

Theorem 1: For some benchmark with helper threads, after the percents of late, timely, bad, ugly, Redundant_MSHR and Redundant_Cache to helper thread LLC prefetching requests are computed respectively, and their distribution has been gotten.

a-1. if the sum of late-percent and timely-percent dominates the distribution, then the performance improvement is true;

a-2. if the percent-sum of bad, ugly, Redundant_MSHR and Redundant_Cache dominates the distribution, then the performance improvement is false;

a-3. if the sum of late-percent and timely-percent is nearly equal to the percent-sum of bad, ugly, Redundant_MSHR and Redundant_Cache, then the performance improvement is uncertain.

III. IMPLEMENTATION

A. Hardware Support

A few hardware components are added in the aforementioned Archimulator simulator to implement the prefetch breakdown:

•LLC Request and Replacement Event Tracking.To monitor the request and replacement activities in the LLC, we need to consider the event when the LLC receives a request coming from the upper level private cache, whether it is a hit or a miss.

•Helper Thread LLC Request State Tracking. In order to track the helper thread request states in the LLC, we need to add one field named threadId to each LLC line to indicate whether the line is brought in by the main thread or the helper thread or otherwise invalid.

•Helper Thread LLC Request Victim State Tracking.In order to track victims replaced by helper thread requests, we need to add an LRU cache named Helper Thread Request Victim Cache (HTRVC) to maintain the LLC lines that are evicted by helper thread requests. The HTRVC has the same structure of the LLC, and there is a direct mapping between the LLC lines and the HTRVC lines. One field called HTRequestTag is thus added to HTRVC to enable reverse lookup in the HTRVC by the helper thread request tag in LLC. The HTRVC has the sole purpose of profiling, thus it has no impact on the program performance.

•Detecting Late Helper Thread LLC Requests.In order to measure late helper thread requests, we need to identify the event when a main thread request hits to an LLC line which is being brought by an in-flight helper thread request coming from the upper level cache. This can be accomplished by monitoring the LLC Miss Status Holding Registers (MSHRs).

B. Tracking and Classifying Helper Thread Prefetches

Changes to the contents of the LLC and HTRVC take place when filling an LLC line and servicing an incoming LLC request. Therefore, corresponding appropriate actions should be taken in the LLC and the HTRVC to track the helper thread LLC requests and victims used in the breakdown of helper thread requests.

•**On Inserting an LLC Line** When filling an LLC line, we should consider four non-trivial cases as listed below: Case 1. A helper thread request fills an INVALID line. In this case, no eviction is needed, but a new NULL entry should be inserted in the HTRVC to maintain the invariant since the data brought by a new helper thread request arrives in the LLC. Case 2. A helper thread request evicts an LLC line which is previously brought in by a main thread request. In this case, a new DATA entry should be inserted in the HTRVC to maintain the invariant. Case 3. A helper thread request evicts an LLC line which is previously brought by a helper thread request. In this case, the stored HT request tag should be updated to the incoming helper thread request tag in the corresponding victim entry in the HTRVC. Case 4. A main thread request evicts an LLC line which is previously brought by a helper thread request. In this case, since the data brought by a helper thread request is removed, its corresponding victim entry found in the HTRVC should be invalidated to maintain the invariant.

•**On Servicing an Incoming LLC Request** When servicing an incoming LLC request, we should consider three nontrivial cases as listed below. The involved LLC line's inflightThreadId is recorded when the incoming LLC request from one thread hits the LLC MSHRs, and its data is being brought in by an earlier LLC request from another thread. Case 1. If the incoming LLC request is from the helper thread, we need to first check the contents of LLC and LLC MSHRs in order to determine if the helper thread request is of either a redundant_mshr or a redundant_cache type. Case2.A bad helper thread request occurs when an incoming main thread LLC request hits a victim entry found in the HTRVC, but misses in the LLC. Case 3. A good helper thread request, either late or timely, depending on whether the value of inflightThreadId is valid or not in the involved LLC line, occurs when an incoming main thread LLC request hits the data brought by a previous helper thread request in the LLC but misses in the HTRVC.

• **On an LLC Line is Invalidated** When an LLC line is invalidated, the involved LLC line's helper thread request state is cleared. If the LLC line is brought by the helper thread, then the corresponding HTRVC

entry is invalidated as well.

IV. EXPERIMENTAL RESULTS

We perform the experiments using selected memory-intensive, pointer traversal benchmarks *mst* and *em3d* from the Olden pointer traversing benchmark suite and *429.mcf* from CPU2006. All applications are cross-compiled to MIPS II executables using gcc flags "-O3". Default values of the helper thread prefetching lookahead and stride parameters are chosen as 20 and 10 respectively, if not specified explicitly. To reduce the simulation time, we first fast forward to the *pthread_spawn(..)* function call for spawning the helper thread in the program code, and then run in detail for 200 million instructions on the main thread. Although there is no *pthread_spawn(..)* call in the baseline program code, we can insert the same pseudo-calls in the corresponding code location as compared to the helper thread prefetching version.

A.Performance Sensitivity to L2 Size and Associativity

Figure2 shows the speedup impact of L2 size and associativity on performance of *mst*,*429.mcf* and *em3d*. The average speedup of *mst*, *429.mcf* and *em3d* is about 1.74, 1.04 and 0.98 respectively. For *mst* of high performance, when the L2 size is greater than 128KB, the execution time remain nearly constant. This is because the average percent of late, timely, bad, ugly, Redundant_MSHR and Redundant_Cache to helper thread LLC prefetching requests is about 78%, 20%, 0, 0.3%, 0.2% and 1.5% respectively, according to theorem1(a-1), the usefulness of late and timely is almost 98%. For *429.mcf*, there is little performance improvement too, and when the L2 size is greater than 128 KB, the execution time reduces little, and the execution times remain nearly constant when the L2 associativity is greater than 8 ways. However, for *em3d*, there is not any speedup. This is because the average percent of late, timely, bad, ugly, Redundant_MSHR and Redundant_Cache to helper thread LLC prefetching requests is about 0, 1.6%, 0, 97%, 0.002% and 1.2% respectively. According to theorem1(a-2), only the ugly is almost 97%, its performance improvement is false.

B.Performance Sensitivity to Helper Thread Parameters

Table1 shows the impact of prefetching parameters such as lookahead and stride on performance and helper thread prefetch breakdown, when L2 Size=1MB, L2 Assoc=8, the *em3d_baseline* execution time is 963314606. Compared em3d (lookahead=20,stride=10) with Figure 2, the minimum reduction of execution time is achieved when the stride is 10 and there is no lookahead. The speedup is changed into 1.23. From the view of breakdown of helper thread L2 requests of *em3d_ht*, Timely helper thread requests were improved distinctly.

From the above discussions of the experimental results on the data intensive, pointer traversal benchmarks from Olden and CPU2006, we can conclude that our performance model of helper thread is effective. Meanwhile the relationship between the lookahead-value and the stride-value reflects the dynamic balance of memory accesses skipped in the prelude and done in stable phase in the helper thread prefetching scheme. For enlarging the cases of timely and late, and reducing the cases of ugly and bad, we must select the optimal values of helper thread parameters such as lookahead and stride by analyzing the memory access patterns of target workload.

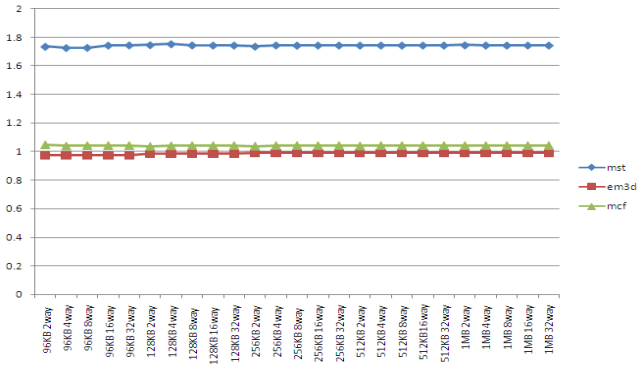

Figure 2. Performance Impact of L2 Size and Associativity

TABLE1. PERFORMANCE IMPACT FOR PARAMETER VALUES OF *em3d_ht*

Stride	*Look-ahead*	*Num_ Cycles*	*Late*	*Timely*	*Bad*	*Ugly*	*Redundant_MSHR*	*Redundant_ Cache*
.10	0	782029586	480	1232504	0	1300084	74	26526
10	10	973337567	73	49317	0	3046216	78	36726
20	0	793020908	688	1165799	0	1413427	69	24536
20	10	973439744	77	48983	0	3060318	84	32904
40	0	818276057	813	1010254	0	1648486	81	24643
40	10	973462738	80	48724	0	3061356	80	32359

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

REFERENCES

[1] S. Byna, Y. Chen, X.-H.Sun,"A taxonomy of data prefetching mechanisms," Proceedings of the International Symposium on Parallel Architectures, Algorithms, and Networks(ISPAN '08), IEEE Computer Society, Washington DC, USA,2008, pp. 19–24.

[2] M. Cai, Archimulator open source cmp architectural simulator. http:// github.com/mcai/Archimulator/.

[3] M. Cai, Z. Gu, "Evaluating the memory system performance of software-initiated inter-core llc prepushing," Proceedings of the 2011 Ninth IEEE International Symposium on Parallel and Distributed Processing with Applications Workshops (ISPAW),2011, pp. 216–221.

[4] J.-C. Chiu, etc., "A unitable computing architecture for chip multiprocessors," The Computer Journal 54 (12), 2033–2052,2011

[5] Z. Gu, Y. Fu, etc., "Improving performance of the irregular data intensive application with small computation workload for CMPs," Proc. ICCP Workshops. IEEE, pp. 279–288,2011.

[6] J. L.Henning, "Spec cpu2006 benchmark descriptions," ACM SIGARCH Computer Architecture News 34 (4), 1–17, 2006.

[7] N.D.E.Jerger, E.L.Hill,M.H. Lipasti,"Friendly fire: understanding the effects of multiprocessor prefetches," ISPASS. IEEE Computer Society, pp. 177–188,2006.

[8] B. Mehta, etc., "Cache showdown: The good, bad and ugly," Tech. rep.2004.

[9] A. Rogers, etc.,"Supporting dynamic data structures on distributed-memory machines," ACM Transactions on Programming Languages and Systems 17 (2), 233–263,1995.

[10] V. Srinivasan, etc.,"A prefetch taxonomy," IEEE Transactions on Computers 53 (2), 126–140,2004.