Prefetching Techniques for STT-RAM based Last-level Cache in CMP Systems*

Mengjie Mao, Guangyu Sun†,Yong Li, Alex K. Jones, Yiran Chen
University of Pittsburgh, Pittsburgh, PA 15261, USA
†CECA, Peking University
{mem231, yol26, akjones, yic52}@pitt.edu, †gsun@pku.edu.cn

***Abstract*—Prefetching is widely used in modern computer systems to mitigate the impact of long memory access latency by paying extra cost in memory and cache accesses. However, the efficacy of prefetching significantly degrades in the memory hierarchy using the emerging *spin-transfer torque random access memory* (STT-RAM) as last-level cache (LLC) due to the long write access latency. In this work, we propose two orthogonal but complimentary techniques to improve the prefetching efficacy of STT-RAM based LLC in chip multi-processor (CMP) systems, namely, request prioritization (RP) and hybrid local-global prefetch control (HLGPC). Simulation results show that by combining these two techniques, we can achieve 6.5%∼11% system performance improvement and 4.8%∼7.3% LLC energy saving in a quadcore system with a 2MB∼8MB STT-RAM based LLC, compared to the system with only basic prefetching.**

I. INTRODUCTION

Prefetching technique [1], [6] is widely adopted in modern computer systems to alleviate the impact of long memory access latency. Data is pre-loaded into a cache based on run-time prediction before it is actually requested. The memory access latency is hidden though extra burdens are imposed on the traffic from the memory to the cache. Potential access conflicts, e.g., bank access contention and cache pollution, may be introduced.

Very recently, *spin-transfer torque random access memory* (STT-RAM) gains increasing attentions in on-chip cache implementation [16]. Compared to SRAM, STT-RAM has nearly zero leakage power consumption, much higher cell density, and similar read performance. However, the long write latency and high write energy primarily impede the adoption of STT-RAM as last-level cache (LLC). Although write performance can be improved by raising write current, the storage density must be sacrificed because a large-size driving transistor is required.

In a chip multi-processor (CMP) system with prefetching, LLC commonly serves prefetch requests (writes) besides common accesses. The introduction of STT-RAM based LLC results in some potential adverse impacts on the system performance and energy: For a LLC built with high-density/small-size STT-RAM cells, the long write access latency increases the possibility of bank access conflicts; for a LLC built with low-density/large-size STT-RAM cells, the limited LLC capacity makes the system more sensitive to the prefetching-incurred cache pollution, causing the increases in cache miss rate and the number of write accesses.

In this work, we propose two orthogonal but complimentary techniques to mitigate the impacts of long write latency of STT-RAM based LLC on the performance of the CMP systems with aggressive prefetching. The first technique is named as *request prioritization* (RP) where different access types of LLC are prioritized based on their criticality to the system performance. Requests with low priority may be preempted by the ones with higher priority during executions. A *hybrid local-global prefetch control* (HLGPC) mechanism is then introduced to dynamically tune the aggressiveness of prefecher for alleviating the LLC access contention.

Compared to the existing works on the memory hierarchy built with STT-RAM, our two major contributions are:

- We quantitatively analyze the degradation of prefetching efficacy in the CMP systems with STT-RAM based LLC.
- Two complimentary techniques – RP and $HLGPC$ are innovated to mitigate the adverse impacts of aggressive prefetching on CMP systems, achieving substantial performance speedup and energy saving under different cache design configurations.

To the best of our knowledge, this is the first work quantitatively analyzing the impact of adopting STT-RAM as LLC on the prefetching efficacy of CMP systems. Simulation results show that the combination of RP and $HLGPC$ techniques can achieve 6.5%∼11% system performance improvement (geometric mean) as well as 4.8%∼7.3% LLC energy saving for a quadcore system with 2MB∼8MB STT-RAM based LLC.

The rest of our paper is organized as follows: Section II gives a review of STT-RAM based LLC design and the recent prefetching efficacy enhancement techniques; Section III depicts the motivations of our work; Section IV shows the implementation details on the RP and $HLGPC$; Section V presents our experiment setup; Section VI shows the simulation results; Section VII concludes our work.

II. BACKGROUND

A. STT-RAM based LLC

The most popular STT-RAM cell design is "1T1J", which contains one magnetic tunneling junction (MTJ) device connected with an access transistor. The resistance of the MTJ can be switched between high- and low-state when a current with different polarization is applied. The MTJ switching can be speeded up by raising the switching current. However, it needs to enlarge the NMOS transistor size and results in the increase of the cell area. Table 1 shows the parameters of three STT-RAM cache designs built with the access transistors with different driving abilities; all parameters are extracted from NVSim [15]. When the MTJ switching time is 3ns, 10ns, and 30ns, the corresponding STT-RAM cell area is about $71F^2$, $25F^2$, and $15F^2$, respectively. Here F is the feature size fabrication technology, say, 45nm here.

TABLE I: The timing and energy parameters of 2/4/8MB STT-RAM LLC at 45nm technology

	2MB			4MB		8MB	
	Write		Read	Write	Read	Write	Read
Timing(ns)	Pulse	3	3.82	10	3.06	30	4.04
	Peripheral	1.69		1.75		2.07	
Cell area(F^2)	71			25		15	
Dynamic energy(nJ)	1.268		0.77	1.08	0.877	1.310	1.197
Memory area(mm^2)	10.429			9.733		10.163	
Leakage power(mW)	924.7			800.4		859.0	

Dong *et al.* first proposed using STT-RAM to implement LLC in microprocessors and discovered its advantages on leakage power reduction [14]. Sun *et al.* extended STT-RAM based LLC design

*This work is supported in part by NSF awards CNS-1116171, NSF of China (No. 61202072), and National High-tech R&D Program of China (No. 2013AA013201).

978-1-4799-2816-3/14/$31.00 ©2014 IEEE

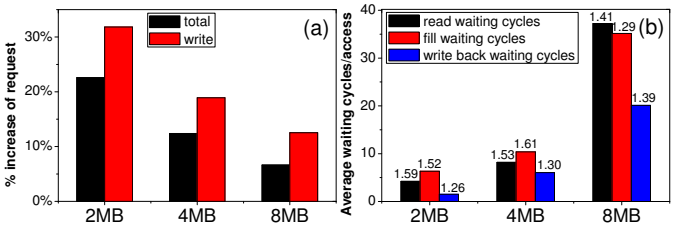

Fig. 1: (a) The increase of total LLC access requests and write requests after applying prefetching; (b) The average waiting cycles of different types of LLC accesses (values atop each bar are normalized to the results without prefetching).

to CMP systems and identified the negative impacts of long write latency of STT-RAM on the system performance [5]. A read-preemptive write buffer, which allows the read request to preempt the write back request that is accessing the same cache bank, was also proposed to alleviate the blocking effect of the write access to STT-RAM based cache. The reliability of STT-RAM read/write operation has been well studied [18], [13], [12] promising the installation of STT-RAM on modern microprocessor design.

B. Prefetching Efficacy Enhancement

The inaccuracy of data prefetcher introduces significant memory resource wasting and degrades system performance. For example, inaccurate prefetch requests incur excessive write accesses to the LLC, which compete with normal LLC requests. As a result, the induced cache pollution and the increased cache miss rate trigger more LLC requests and further aggravate the competition among the requests. In [11], [17], filter techniques are proposed to dynamically tune the prefetch requests based on the prefetech accuracy while a hardware-based cache pollution filter is constructed for the prefetcher. If the predicted accuracy of a particular prefetch request is lower than a preset threshold, the corresponding request will be prevented from being issued. In [2], [9], the aggressiveness of prefetchers, i.e., the prefetch degree[1] and prefetch distance[2], is dynamically adjusted based on the real-time system feedback information like temporal accuracy, prefetch timeliness, cache pollution, and memory bandwidth utilization. These techniques are traditionally applied to shared main memory in the presence of multiple prefetchers.

III. Technical Motivation

Figure 1(a) shows the increases in the total LLC access requests and the write requests after applying prefetching to a STT-RAM based LLC with a capacity of 2MB, 4MB, and 8MB, respectively, compared to the case that no prefetching is applied. The data is averaged over six workloads, each of which consists of 4 different applications running simultaneously on a quadcore CMP (more simulation setup details will be given in Section V). As shown in Table I, all LLC configurations have the similar areas ($\sim 10mm^2$) but different performances: large cache capacity is associated with long write access latency due to the small transistor size. The shortest read latency, which is the sum of delays on memory array and peripheral circuits, happens in the 4MB LLC design. Figure 1(b) depicts the average waiting cycles of each types of LLC access requests caused by cache bank access conflicts after prefetching is applied. Based on Figure 1(a) and (b), we made the following observations:

- With prefetching, the numbers of total access requests as well as write requests climb up, followed by the increase in the waiting time for all types of access requests. As the capacity of LLC increases, the average waiting time of all types of access requests keeps raising due to the prolonged write access latency. Figure 1 (b) also show that read requests and fill requests, which are commonly located on the critical path of execution, experience much longer waiting time compared to the less critical requests, e.g., write back. Therefore, the system performance may be potentially improved by assigning higher priority to the critical LLC access requests during execution to avoid long waiting time.

- The prefetching efficiency is affected by the capacity (cell size) of STT-RAM based LLC. A large cell size helps to shorten the average waiting time of the LLC access requests by reducing the blocking time of write operations. However, the cache pollution incurred by prefetching also becomes severer due to the reduced total capacity. On the contrary, the average waiting time of the LLC access requests rises when the LLC capacity increases. The system performance is determined by the tradeoffs among the write request frequency, the average waiting time (or the LLC access conflicts), and the prefetching policy, all of which vary during the execution of a program. Hence, we may dynamically throttle the aggressiveness of prefetchers for system performance improvement by taking into account all these factors.

[1]# of prefetch requests that can be issued in one access.

[2]The range that the prefetch requests locate into.

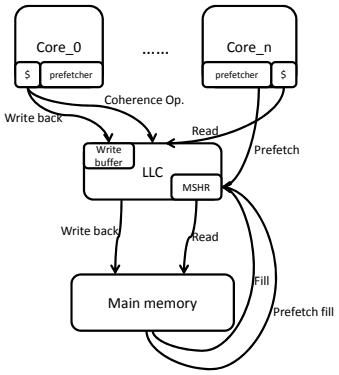

Fig. 2: Access types of the shared LLC in CMP systems.

IV. Proposed Techniques

A. Request Prioritization (RP)

Figure 2 summarizes the common LLC access requests in a CMP system: read requests, coherence requests[3] and write back requests are initiated by the upper-level cache of each CPU core; prefetcher sends the prefetch requests to the main memroy and fills the corresponding data into the LLC. If the read request to the LLC encounters a cache miss, a fill request will be generated to fetch the data from the main memory to the LLC. Consequently, the following accesses to the corresponding cache bank of the STT-RAM based LLC may be blocked due to long write latency of memory cells, leading to a bank access conflict.

Notice that these LLC access requests shown in Figure 2 demonstrate different criticality to the system performance. For example, read requests must be handled immediately as the cache miss at upper-level cache significantly affects the system performance. Fill requests are also critical since they are triggered by LLC misses, especially if the requested data is loaded into LLC first before it is sent to upper-level cache for consistency. Prefetch fill requests, however, have lower criticality because the prefetched data is unlikely to be used immediately. Write back requests apparently have the lowest criticality because the data written back to the LLC may not be used in the near future.

[3]Most cache coherence protocols only modify the states stored in the tag array/directory that is highly banked, without accessing the data array. Thus, coherence request is not included in our RP technique.

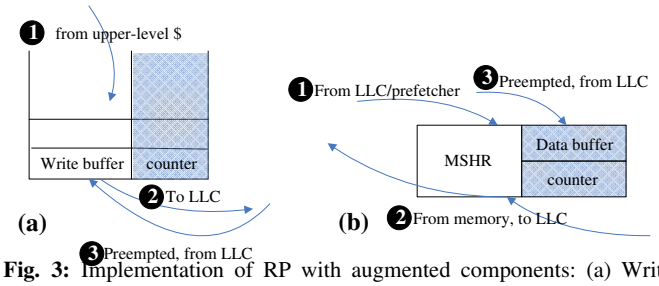

Fig. 3: Implementation of RP with augmented components: (a) Write back requests. (b) Fill and prefetch fill requests.

We proposed to prioritize the different types of LLC access requests based on their criticality to the system performance, and process the requests based on their priorities. For example, if multiple requests compete the access to the same cache bank, the one with the highest priority will be granted the right of access. However, a request with a high priority may also be blocked by a low priority request which is being processed. To address this issue, we allow the high-priority request to preempt the low-priority one if the elapse time of processing the low-priority request is below a threshold, say, the retirement accomplishment degree (RAD) [5]. For example, a 30% RAD of write back request means that the write back request can be preempted by a request with higher priority if the elapse time of the LLC write back access is below 30% of the STT-RAM write latency. We refer to this method as *request prioritization* (RP).

To record the elapse time of the LLC access request, each *miss state handling register* (MSHR) or write buffer entry must be augmented with a N-bit counter (e.g., 7-bit to record up to 128-cycle operation time for an 8MB LLC). The counter is initialized once the associated request grasps the access right of the cache bank, and reset after the request is preempted or successfully processed. As shown in Figure 3(a), when a write back request is preempted, the data being written is still retained at the head of the write buffer and waiting for the next try; otherwise, the buffer entry will be recycled once the write back completes.

However, if a fill or prefetch fill request is preempted, the request must be buffered for the future retry. As shown in Figure 3(b), besides the elapse time counter, a data buffer whose length is the same as one cache block is also augmented to the MSHR to buffer the data associated with the fill/prefetech fill request. The procedure which is similar to LLC miss handling can be applied to deal with the write preemption: Once the request is issued, a MSHR is assigned to it by LLC controller; the MSHR holds the data fetched from the main memory until it is successfully written into the LLC; if the request is preempted before the writing is completed, the request is buffered in the MSHR and waiting for the next retry; otherwise, the MSHR will be recycled after the writing is completed. Read requests will never be preempted.

Preempting write request may cause false LLC states since the tag array would be updated before the data array is updated. For instance, a write request to the data array is preempted after modifying the valid/dirty bits of the corresponding tag entry. If a read request accesses the same data entry before the retry of the write request, it may actually access the old or invalid data. To avoid this inconsistency, we define the policy that a write request cannot update the valid bit in the tag entry until it successfully updates the data array. The read request in the above example will trigger a cache miss which will be filtered by the MSHR entry occupied by the preempted write request.

Similar to the scheme in [5], a 20-entry write buffer is adopted in our design. To suppress the competition between the prefetch fill requests and fill requests and avoid overflow, the number of MSHRs must be sufficiently large, e.g., 128 in our simulated quadcare system. The augmented storage is built with SRAM.

TABLE II: GPC decision guideline

Case	$core_i$'s info		LLC's info	Decision
	Pref. Acc.	Pref. Freq.	Acc. Freq.	
1	High	Low	High	Allow local
2	High	High	Low	Allow local
3	High	High	High	Disable scale up
4	High	Low	Low	Allow local
5	Low	Low	High	Disable scale up
6	Low	High	Low	Allow local
7	Low	High	High	Force scale down
8	Low	Low	Low	Allow local

B. Hybrid Local-global Prefetch Ctrl (HLGPC)

As aforementioned in Section III, overaggressive prefetching in a CMP system may cause cache pollution and aggravate the access conflict of the shared memory resource [4]. This issue becomes severer in the system with small LLC because prefetch fill requests will have a higher probability to evict the useful cache blocks. The increased LLC miss rate will generate more prefetches and eventually put the system into a negative feedback loop. The general solutions of this issue includes *local (per core) prefetch control* (LPC) [3], [9] and prefetcher coordination by considering main memory access contention [2], [4], which dynamically control the aggressiveness of the prefetchers.

However, conventional LPC technique focuses on maximizing the performance of each CPU core, which does not necessarily optimize the overall system performance due to the LLC access conflicts among the different prefetchers. Also, the significant access conflicts in STT-RAM based LLC become another crucial factor affecting the prefetching efficacy besides main memory access contention. Based on the above observations, we propose *hybrid local-global prefetch control* (HLGPC) technique to achieve a balanced dynamic aggressiveness control across all prefetchers in a CMP system. The two integrated components of HLGPC are:

Local prefetch control (LPC): At core-level, we choose *feedback directed prefetching* (FDP) [9] as the LPC scheme to manage the aggressiveness of the prefetcher. FDP periodically samples *the prefetcher's accuracy*, *prefetch timeliness* and *per-core-induced cache pollution* over every time interval and determines the aggressiveness of the prefetcher in the next interval accordingly.

Global prefetch control (GPC): At chip-level, GPC may retain or override the decision of LPC based on the runtime information of LLC. GPC periodically samples the *prefetch frequency* of each core and the *global access frequency* of LLC. Based on the values of these two metrics and the prefetcher's accuracy over the current interval, GPC applies the following three rules on the prefetcher of each core in the next interval, as shown in Table II.

- Disable scale up: In case 3, although the prefetch accuracy and frequency are high for $core_i$, the high global access frequency of the LLC indicates the severe access conflicts. Therefore, the scaling up of the prefetcher's aggressiveness is disabled even the LPC decides to do so. In case 5, continue scaling up the prefetcher's aggressiveness in $core_i$ will likely deteriorate access conflicts due to high LLC access frequency and low prefetch accuracy. Hence, the scaling up of the prefetcher's aggressiveness is also prevented;
- Force scale down: In case 7, low prefetch accuracy couples with high prefetch frequency, indicating high volume of useless prefetches. Considering the high global access frequency of the LLC (or say, the severe access conflicts), GPC will force $core_i$ to scale down the prefetcher's aggressiveness regardless the decision of LPC;
- Allow local decision: For the rest cases in Table II, GPC will stick to the decision of LPC which ensures the best performance

TABLE III: The aggressiveness levels of prefetchers

Aggressiveness	Prefetch Distance	Prefetch Degree
Very conservative	4	1
Conservative	8	2
Medium	16	4
Aggressive	32	4
Very aggressive	64	8

for each core as well as the overall system.

In our implementation of HLGPC, the design metrics like prefetch frequency of $core_i$ and the global access frequency over interval i are updated as [9]:

$$UpdatedCount_i = 0.5 \cdot UpdatedCount_{i-1} + 0.5 \cdot CurrentCount_i. \quad (1)$$

Here $UpdatedCount_i$ is the metric over the current interval i while $UpdatedCount_{i-1}$ is the metric over the previous interval $i-1$. $CurrentCount_i$ denotes the number of the prefetch requests issued by $core_i$ or the LLC accesses over the current interval i. Eq. (1) gives the credits to the metrics over both the current and all previous intervals and reflects the temporal correlation of the execution of the workload. The threshold value of each GPC metric adopted in our simulation will be shown in Section V-A.

V. Experiment Setup

A. Simulation Platform

Without loss of generality, we adopt the stream prefetcher in IBM POWER4 processor [6] in our simulation among the exisiting prefetcher designs. The aggressiveness of stream prefetcher can be dynamically adjusted among five configurations, as shown in Table III. We also assume there is only one prefetcher per CPU core in the simulated CMP system.

The MacSim [19] simulator with Intel's Sandy Bridge configuration is used in our evaluation. The design parameters of STT-RAM based LLC at 45nm technology, which are summarized in Table I, are obtained from NVsim [15] with the appropriate device parameters in [7]. We also consider the energy overhead of the augmented write buffer and MSHR while the energy parameters of these components are extracted from CACTI [10] and Synopsys using VHDL. The configuration of the simulated quadcore system is summarized in Table IV. The length of an interval in the HLGPC scheme is set to 8192 evictions from LLC [9]. The threshold values for the high/low prefetch accuracy, the prefetch frequency, and the global access frequency are empirically set to 0.5, 246 and 8192, respectively. Total four 13-bit counters are needed in each core to measure the prefetch accuracy, the prefetch frequency, and the global access frequency of LLC (it needs two counters).

B. Selection of Benchmarks

We select 18 benchmarks from SPEC CPU 2000/2006 suite. The execution trace of each benchmark is collected by using a modified pin-based trace extraction tool packed with MacSim tool set. Each trace is composed of 400 million instructions after bypassing the initialization phase ranging from 5 billion to 100 billion instructions.

We construct 6 multi-programmed workloads which generally suffer from LLC conflicts with prefetching. Each workload consists of 4 benchmarks, among which at least two benchmarks are *LLC access intensive* (LLCI), one benchmark is *LLC non-intensive* (LLCNI), and one benchmark is *prefetch intensive* (PREFI). Here, LLCI is defined as the *accesses per 1 kilo instructions* (APKI) of LLC > 8; LLCNI is defined as APKI ≤ 8; and PREFI is defined as the *prefetch requests per 1 kilo instructions* (PPKI) of LLC > 1. The workloads that every benchmark belongs to and their properties are summarized in Table V. These properties are profiled by running each benchmark on only one core of a quadcore system with prefetching.

TABLE V: Properties of 18 SPEC benchmarks

benchmark	LLCI	PREFI	Workload
art	Y	Y	1, 4
ammp	Y	N	3
bzip2	N	N	1
gcc	N	N	6
gobmk	N	N	6
milc	Y	N	1
zeusmp	Y	Y	4, 6
gromacs	N	N	5
cactusADM	N	N	2
leslie3d	Y	Y	1, 2
dealII	N	N	2
hmmer	N	N	3
GemsFDTD	Y	Y	3
h264ref	N	N	5
lbm	Y	Y	2, 4
omnetpp	Y	Y	3, 5
astar	N	N	4
xalancbmk	Y	Y	5, 6

VI. Results and Discussions

A. The Efficacy of RP

Table VI depicts three priority assignment schemes evaluated in our simulations, namely, P1, P2, and P3. The priorities among different requests under different priority assignments are represented by RAD. For example, read has a higher priority than fill (25% RAD under P2), which means that fill will be preempted by read if its elapse time is below 25% of the write access latency of the LLC. We conducted extensive experiments to explore the performance/energy impact of different RAD settings and extracted three representative priority assignments, as summarized in Table VI.

Figure 4 depicts the *Harmonic mean speedups* (HS) [8] of every workloads under different RP schemes (P1, P2, P3), respectively. The results are normalized to the HS of the same system without prefetching. For comparison purpose, we also provide the results of conventional prefetching technique (*pref. only*) and the geometric mean of the HS of the read-preemptive write buffer technique [5] (RAD_{org}). The results show that prioritizing LLC access requests always achieve system performance improvement w.r.t. conventional prefetching technique though the results of different RP schemes vary at different workloads and LLC configurations. The highest performance is achieved by P1 with a HS improvement of 4.6%/4.8%/8.3% for 2/4/8MB LLC compared to *pref. only*. RAD_{org}, however, receives very marginal performance improvement across all scenarios as it only differentiates the priorities between read and write back requests.

Figure 4 also shows that compared to no prefetching, conventional prefetching technique achieves marginal or even negative performance improvement (i.e., wl2) in all workloads. This fact clearly reflects the adverse impacts induced by the LLC access conflicts due to the long write access latency of STT-RAM. The detailed analysis on the efficacy of all RP schemes are:

- Among all RP schemes, P1 achieves the best performance because the critical requests like read and fill are always served in a timely manner;
- Comparing P1 and P2, lifting up the priority of fill request (P1) results in performance improvement, indicating that the criticality of read and fill requests are equivalent or at least close to each other in the simulated workloads;

TABLE VI: Priority assignment schemes

	read			fill			prefetch fill			write back		
	P1	P2	P3	P1	P2	P3	P1	P2	P3	P1	P2	P3
read	-	-	-	-	25%	100%	60%	60%	100%	60%	60%	100%
fill	-	-	-	-	-	-	60%	60%	60%	60%	60%	60%
prefetch fill	-	-	-	-	-	-	-	-	-	60%	60%	60%
write back	-	-	-	-	-	-	-	-	-	-	-	-

TABLE IV: System Configuration

Execution	4GHz, OOO 4 issues(up to 2 mem and 2 FP), 20 stages pipeline, gshare branch predictor, 1024-entry BTB, global history length 14, 20 cycles branch misprediction penalty, 256-enrty ROB
Upper-level cache	32KB L1IC, 4-way, 4 banks, 4 ports, 1-cycle latency, 64B line 32 KB L1DC, 4-way, 4 banks, 4 ports, 2-cycle latency, write back, 64B line 256 KB L2, 8-way, 4 banks, 1 r/w port, 8-cycle latency, write back, 64B line, 64 MSHRs
Prefetcher	Stream prefetcher with 64 streams, max prefetch degree 8, max prefetch distance 64
LLC	2/4/8 MB STT-RAM shared L3, 8/16/16-way, 8/8/8 banks, 1 r/w port, 16/12/16-cycle read, 20/48/128-cycle write, 64B line, write back, 128 MSHRs
Main memory	Fixed 200-cycle latency

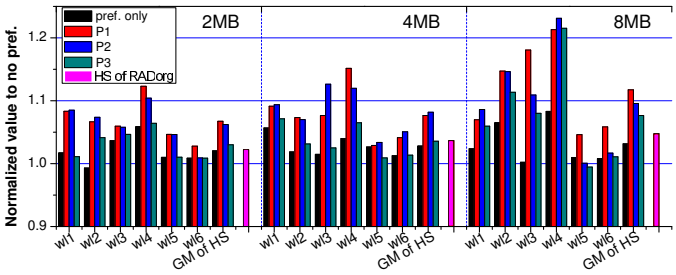

Fig. 4: HS of *perf. only* and different RP schemes. Geometric means of the HS of RAD_{org} are also included.

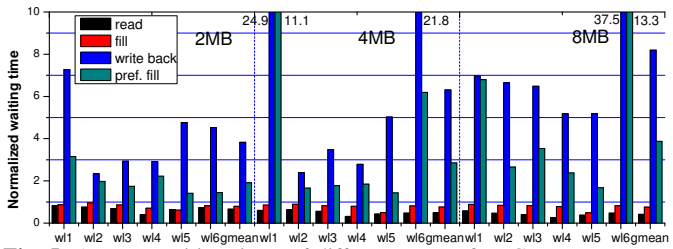

Fig. 5: Average waiting times of different types of LLC access requests of P1.

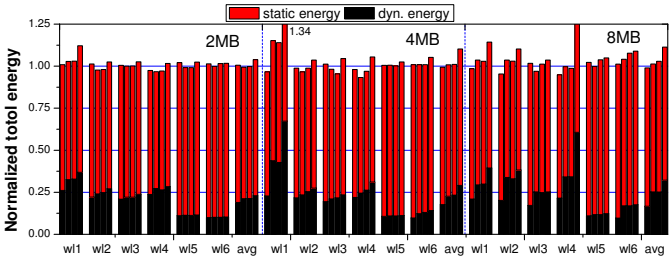

Fig. 6: LLC energy consumption of different prefetching schemes. From left to right for each workload: *pref. only*, P1, P2 and P3.

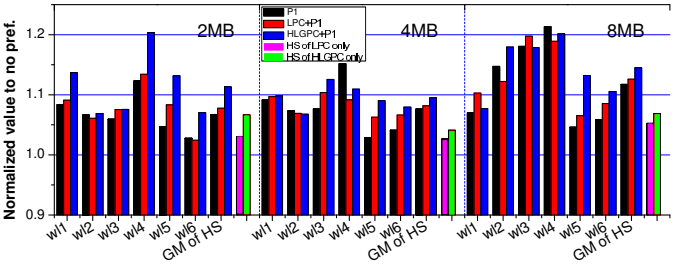

Fig. 7: HS of P1, LPC+P1 and HLGPC+P1. Geometric means of the HS of *LPC only* and *HLGPC only* are also included.

- Comparing P2 and P3, the aggressive priority assignment to read request (P3) introduces very little performance improvement. It is because of the overflowing of the write buffer incurred by the frequent preemption from the read requests as well as the equivalent criticality of read and fill requests.
- In some cases (e.g., wl3 under 4MB, wl1 and wl4 under 8MB), system performance is more sensitive to the priority assignment of read requests; thus, P2 noticeably outperforms P1 for those cases.
- RP scheme achieves more substantial performance improvement for large LLC with long write access latency than small LLC because of the severer LLC bank conflict. For an 8MB LLC, the geometric mean of HS for P1 can be as high as 8.3%.

To gain the insight into the efficacy of P1, Figure 5 lists the average waiting times of different types of LLC access requests of P1, which are normalized to the result of *pref. only*. The results show that most of the waiting time has been allocated from read and fill requests to prefetch fill and write back requests. Less waiting time of the critical requests is the main reason for system performance improvement.

Figure 6 summarizes the LLC energy consumptions of the different prefetching schemes that are normalized to no prefetching. The extra write operations induced by the preemptions in RP schemes raise the dynamic energy of LLC. Since P1 does not preempt fill requests, it consumes the lowest dynamic energy among all RP schemes. Because all types of write accesses can be preempted by read requests (i.e., RAD = 100%), the dynamic energy consumption of P3 is significantly higher than any other schemes. Following the increase of LLC capacity, the probability and the number of the low-priority requests being preempted climb up due to the prolonged write access latency, resulting in dynamic energy increase. Luckily, the leakage energy of the LLC is substantially reduced as the execution time decreases. The average LLC energy consumptions increase less than 1% across all LLC configurations. Since P1 achieves well-balanced tradeoff between system performance and energy consumption, we select it as our baseline RP scheme in the evaluation of HLGPC.

B. The Efficacy of HLGPC

Figure 7 depicts the HS for every workload before and after applying LPC+P1 and HLGPC+P1, respectively. The selected RP scheme is P1 and the selected LPC scheme is FDP [9]. The geometry means of the HS of *LPC only* and *HLGPC only* are also included to illustrate the result of the prefetch control without RP scheme. All results are normalized to no prefetching. The performance of *LPC only* is visibly worse than that of *HLGPC only*, indicating the necessity of combining both global and local prefetch control. Compared to applying only P1, both LPC+P1 and HLGPC+P1 improve the average system performance while HLGPC+P1 achieves the highest improvement (4.3%/1.7%/2.5% HS for 2/4/8MB LLC compared to P1 or 9.6%/6.5%/11% HS for 2/4/8MB LLC compared to *pref. only*). Small LLC benefits the most from HLGPC+P1 as the corresponding LLC access conflicts are effectively alleviated.

In HLGPC+P1, the rules applied at GPC level may mitigate the prefetcher's aggressiveness and cause the increase in LLC miss rate. If system performance is more sensitive to LLC miss rate than access conflicts, LPC+P1 could outperform HLPGC+P1, as shown in "wl1" and "wl3" under 8MB LLC. In "wl2" under 4MB LLC, the LLC access conflicts induced by prefetching is very insignificant. Applying extra control on prefetch in LPC+P1 or HLGPC+P1 indeed degrades the system performance compared to applying no prefetch control (P1).

Figure 8 presents the profile of the number of fill and prefetch fill requests and the average waiting times of read and fill requests before

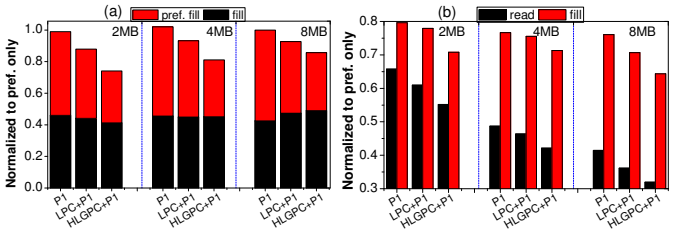

Fig. 8: The normalized (a) Number of fill and prefetch fill requests of P1, LPC+P1 and HLGPC+P1; (b) Average waiting times of critical requests of P1, LPC+P1 and HLGPC+P1.

and after applying different prefetch controls. All data are normalized to *pref. only*. As shown in Figure 8(a), HLGPC+P1 substantially suppresses prefetch fill requests without significantly increasing LLC miss rate denoted by the number of fill requests. It also implies that the miss rate may decrease with less aggressive prefetching because of the alleviation of cache pollution. As shown in Figure 8(b), the average waiting time of read and fill requests for each scheme is also significantly reduced due to the alleviated LLC access conflicts.

Figure 9 summarizes the LLC energy consumptions of different prefetch controls that are normalized to no prefetching. By throttling the prefetch requests, the dynamic energies of both LPC+P1 and HLGPC+P1 are all below that of P1 while HLGPC+P1 achieves the lowest dynamic energy consumption. Nonetheless, as the system performance improves, the total energy efficiency of the LLC with HLGPC+P1 is enhanced by 7.3%/4.8%/5.6% for 2MB/4MB/8MB LLC compared to *pref. only*.

To gain the insight into how HLGPC+P1 globally balances the aggressiveness of different prefetchers in the CMP system, we conducted a case study on the execution of four applications of "wl4" under a 2MB LLC, as shown in Figure 10. Figure 10(a) shows the individual application performance, which is normalized to that when only one application running exclusively on only one core of the quadcore system. After applying P1, the performances of `art` and `lbm` are substantially improved while that of the other two benchmarks are only slightly improved. After introducing LPC+P1, the performance of `art` and `astar` are further improved but those of `lbm` and `zeusmp` slow down a little bit. The microscope analysis in Figure 10(b) shows that after applying LPC+P1, `art` and `zeusmp` experience noticeable prefetch de-aggressiveness that relieve the LLC access conflicts. Simulations also show that the prefetch accuracy in these two applications are low, i.e., 34% for `art` and 46% for `zeusmp` in LPC+P1, while the volumes of prefetch requests are high. Therefore, HLGPC+P1 is able to improve the system performance by