# ABS: A Low-Cost Adaptive Controller for Prefetching in a Banked Shared Last-Level Cache

JORGE ALBERICIO, RUBÉN GRAN, PABLO IBÁÑEZ, and VÍCTOR VIÑALS,
University of Zaragoza
JOSE MARÍA LLABERÍA, UPC Barcelona Tech

Hardware data prefetch is a very well known technique for hiding memory latencies. However, in a multicore system fitted with a shared Last-Level Cache (LLC), prefetch induced by a core consumes common resources such as shared cache space and main memory bandwidth. This may degrade the performance of other cores and even the overall system performance unless the prefetch aggressiveness of each core is controlled from a system standpoint. On the other hand, LLCs in commercial chip multiprocessors are more and more frequently organized in independent banks. In this contribution, we target for the first time prefetch in a banked LLC organization and propose ABS, a low-cost controller with a hill-climbing approach that runs stand-alone at each LLC bank without requiring inter-bank communication. Using multiprogrammed SPEC2K6 workloads, our analysis shows that the mechanism improves both user-oriented metrics (Harmonic Mean of Speedups by 27% and Fairness by 11%) and system-oriented metrics (Weighted Speedup increases 22% and Memory Bandwidth Consumption decreases 14%) over an eight-core baseline system that uses aggressive sequential prefetch with a fixed degree. Similar conclusions can be drawn by varying the number of cores or the LLC size, when running parallel applications, or when other prefetch engines are controlled.

Categories and Subject Descriptors: B.3.0 [**Memory structures**]: General

General Terms: Design, Performance

Additional Key Words and Phrases: Prefetch, shared resources management

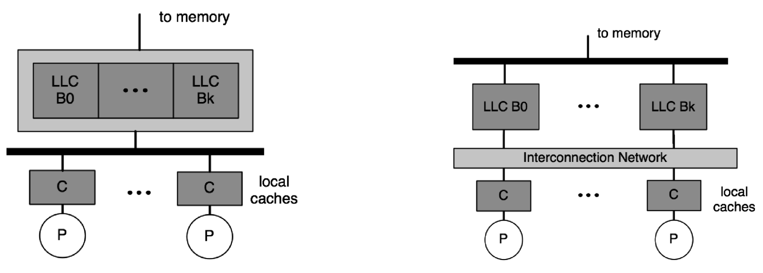
**ACM Reference Format:**
Albericio, J., Gran, R., Ibáñez, P., Viñals, V., and Llabería, J.M. 2012. ABS: A low-cost adaptive controller for prefetching in a banked shared last-level cache. ACM Trans. Architec. Code Optim. 8, 4, Article 19 (January 2012), 20 pages.
DOI = 10.1145/2086696.2086698 http://doi.acm.org/10.1145/2086696.2086698

## 1. INTRODUCTION

Hardware data prefetch is a very well known technique for hiding the long latencies involved in off-chip accesses. It tries to predict memory addresses in advance, requesting them to the next level, and loading the lines into the cache before the actual demands take place. Several commercial multicore processors implement some form of hardware data prefetch [Conway et al. 2010; Le et al. 2007]. These devices are fitted with a hierarchy of cache memories. Usually, the first levels are private and the Last Level Cache (LLC) is shared by all the cores in the system. In this work we consider this kind of system.

This work was supported by grants TIN2007-60625 and TIN2010-21291-C02-01 (Spanish Government and European ERDF), gaZ: T48 research group (Aragon Government and European ESF), Consolider CSD2007-00050 (Spanish Government), and HiPEAC-2 NoE (European FP7/ICT 217068).
Authors' addresses: J. Albericio, R. Gran, P. Ibáñez, and V. Viñals, University of Zaragoza, Dpto. de Ing. de Sistemas e Informática; J. M. Llabería, DAC UPC Barcelona Tech; email: jalberic@unizar.es.

DOI 10.1145/2086696.2086698 http://doi.acm.org/10.1145/2086696.2086698



(a) Monolothic (b) Banked

Fig. 1. Last Level Cache organizations.

Prefetches may be initiated in the first-level caches or directly from events occurring in the LLC, but in the end the prefetches reach the shared LLC and interfere with each other. That is, prefetches issued on behalf of one core may evict LLC lines previously allocated by other cores, either by a memory instruction or a prefetch request. In addition, the prefetch activity originated from a single core can reduce the overall available bandwidth, potentially increasing the latency seen by the demands or prefetches coming from the rest of cores.

Most prefetch proposals in multiprocessors deal with systems having only private caches [Cantin et al. 2006; Dahlgren et al. 1993; Koppelman 2000; Somogyi et al. 2009; Tcheun et al. 1997], while prefetch for shared caches has received little attention [Ebrahimi et al. 2009].

Ebrahimi et al. [2009] tackle for the first time the problem of reducing the prefetch inter-core interference in a chip multiprocessor with a shared LLC. They propose the *Hierarchical Prefetcher Aggressiveness Control (HPAC)* mechanism, that monitors several global indexes (prefetch accuracy, inter-core pollution, and memory controller activities) to adjust the prefetch aggressiveness of each core. This assumes a centralized implementation of the LLC, internally organized in banks but with a single access port (see Figure 1(a)). Thus, the global aggressiveness control and associated hardware structures are also centralized.

To the best of our knowledge, this is the first work where prefetch is studied in a multicore system fitted with a *banked* shared LLC, see Figure 1b. We assume an LLC organized in independent cache banks with an access port each, and an interconnection network attaching cores to cache banks (a crossbar is assumed, but other topologies can be considered) [Kongetira et al. 2005; Kottapalli and Baxter 2009]. Each bank is internally sub-banked in order to provide a higher throughput. We think this kind of LLC will become mainstream in the short term, because independent banks add layout flexibility and increase access bandwidth. Commercial processors from AMD, Sun, Intel, or IBM are using this design [Conway et al. 2010; Kongetira et al. 2005; Kottapalli and Baxter 2009; Le et al. 2007].

In this scenario, we introduce the *ABS controller*, an Adaptive controller for prefetch in a Banked Shared LLC. ABS controllers are installed in all the LLC banks which are already fitted with a prefetch engine. Each ABS controller runs autonomously and gather local statistics to set the prefetch aggressiveness for each core in the bank it controls in order to maximize the *overall* system performance using a hill-climbing approach. Therefore, a given core is allowed to prefetch with different aggressiveness on different banks of the LLC.

Isolation between banks is a key factor of our proposal, meaning that both the ABS controller and the prefetcher in a bank are not influenced by their peers at other banks. Bank isolation achieves two essential benefits, namely (i) the prefetches generated from

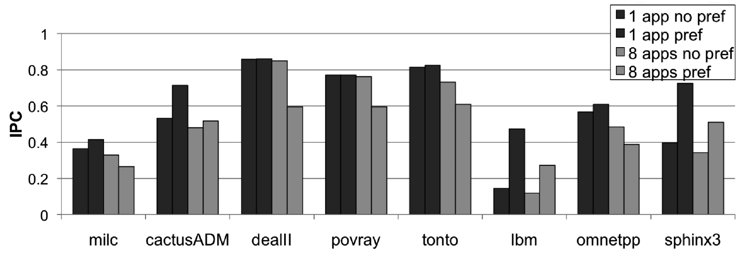


Fig. 2. IPC for eight SPEC2K6 applications (mix2) running on an 8-core system with a shared LLC.

a given bank target itself, and it will always be possible to filter useless prefetches by looking up in the bank, thus saving memory bandwidth, and (ii) communicating prefetchers or ABS controllers among banks is not required, removing the need for a dedicated interconnection network or extra traffic in the existing one. As discussed in Section 5.2, bank isolation can be achieved by selecting a proper address interleaving among banks or by adjusting the prefetch distance.

Our results show that an eight-core system with ABS controllers running multiprogrammed SPEC2K6 workloads improves in both user-oriented metrics and system-oriented metrics over a baseline system with a fixed degree sequential prefetch. The results are consistent when varying the number of cores or LLC sizes. ABS control can be applied to other prefetch engines as long as they are able to operate at different aggressiveness levels. Specifically, we introduce ABS-controlled sequential streams. A comparison with HPAC-controlled sequential streams, such as that proposed by Ebrahimi et al. [2009], shows higher performance at a very small fraction of the cost. Furthermore, when running multithreaded workloads from *SPLASH-2* [Woo et al. 1995] and *PARSEC* [Bienia 2011], the ABS controllers also reduce the execution time and the consumed bandwidth over the baseline system.

The paper is structured as follows. Section 2 describes the motivation behind the work. Section 3 gives some background and reviews related work. Section 4 introduces the ABS controller. Section 5 presents the prefetch framework. Section 6 shows the methodology followed. Section 7 shows the results when ABS controllers are evaluated in a variety of situations, and Section 8 concludes the paper.

## 2. MOTIVATION

Figure 2 shows instructions per cycle (IPC) for eight SPEC2K6 applications running on a system with eight cores and a 4MB shared LLC. The simulation details are shown in Section 6. The figure shows four bars for each application. The first two bars represent programs running alone in the system, either without prefetch or with an aggressive (degree 16) sequential tagged prefetch (see Section 3). The last two bars represent the eight applications running together, either all without prefetch or all with the former aggressive prefetch turned on. When comparing the systems with prefetch (second and fourth bars), significant performance losses appear when resources are shared among cores. Note that prefetch involves virtually no performance loss in any application when running alone (first and second bars), while it causes losses in 5 out of 8 applications when running all together (third and fourth bars). Therefore, in order to boost the shared LLC performance by means of prefetch, a mechanism to control aggressiveness is called for. Such a mechanism should consider global metrics to realize when the prefetch activity of a core harms the overall system performance and it should be decreased in spite of the improvement achieved by that core.

As Figure 2 highlights, the benefit obtained by an application due to prefetch can decrease or even turn into losses if prefetch is simultaneously active in all cores.

Therefore, our goal is to design a mechanism that dynamically controls the prefetch aggressiveness of each core in order to maximize system performance.

The impact on system performance can be assessed using global indexes such as aggregated IPC or shared LLC miss ratio. Both are obtained by adding quantities that are distributed in the cores or the cache banks, respectively. So, a centralized design of the prefetch aggressiveness control requires sending information from the places where events are counted to the centralized control point. Alternatively, we propose to place an ABS controller in each LLC bank. The controller uses bank-local information (i.e. bank miss ratio) to improve bank performance. Improving the performance of every bank will thus improve the system performance.

## 3. BACKGROUND AND RELATED WORK

*Prefetch Aggressiveness.* Prefetch aggressiveness is often defined in terms of degree and/or distance. Let us consider a stream of references a processor is going to demand $(a_i, a_{i+1}, a_{i+2}, \ldots)$, where address $a_i$ has just been issued. A prefetcher can be designed to produce the next $k$ addresses following $a_i$ $(a_{i+1}, \ldots a_{i+k})$, calling $k$ the prefetch degree. Alternatively, or in addition, it can also be designed to produce a single address of a far reference $(a_{i+d})$, calling $d$ the prefetch distance. As an example, we recall the sequential tagged prefetcher with degree $k$ [Dahlgren et al. 1993]. If reference $a_i$ having the line address $V$ is going to trigger a burst of prefetches ($a_i$ misses or is the first use of a prefetched line), then the prefetcher will issue the following request burst $(V+1, V+2, \ldots, V+k)$. Another example of a prefetch engine using aggressiveness is the sequential streams as presented in [Srinath et al. 2007]. In that work, aggressiveness was defined as a combination of distance and degree.

*Mechanisms to Adjust Prefetch Aggressiveness in Multiprocessors.* Several works address the problem of adjusting prefetch aggressiveness in CC-NUMA multiprocessors having only private cache memories. Dahlgren et al. [1993] suggest determining the prefetch aggressiveness at each private cache by counting the number of useful prefetches for every given number of prefetches issued(an epoch); the prefetch aggressiveness is increased or decreased taking into account two usefulness thresholds. Tcheun et al. [1997] add a degree selector to a sequential prefetch scheme; when the selector detects useful prefetches along a sequential sub-stream it increases the prefetch aggressiveness of the next sub-stream belonging to the same stream. As these works do not use a shared cache, the interference problems found among cores are only related with the available bandwidth when accessing to memory.

To our knowledge, only the Hierarchical Prefetcher Aggressiveness Control (HPAC) presented by Ebrahimi et al. [2009] has faced the problem of adjusting prefetch aggressiveness on a shared LLC (HPAC). We notice four main differences between that work and the present one. (i) While HPAC resorts on computing the prefetching aggressiveness by means of a set of rules applied to several system variables (a kind of fuzzy controller), ABS relies on a local search method (a variant of hill-climbing) to minimize a single system variable, the bank miss ratio. (ii) HPAC was proposed for a centralized LLC with a single access port (see Figure 1(a)). We propose ABS for a cache organized in banks, each one with an access port (see Figure 1(b)). (iii) HPAC throttles auto-regulated prefetch engines attached to each core. In the original paper, HPAC is evaluated using Feedback Directed Prefetching as the auto-regulated prefetch engine [Srinath et al. 2007]. However, ABS controllers set directly the aggressiveness level of the local prefetchers. (iv) HPAC uses four global metrics and FDP (as part of HPAC) uses three more local metrics. All these metrics are monitored and compared to ten thresholds (4 for HPAC + 6 for FDP). In contrast, ABS only samples two system

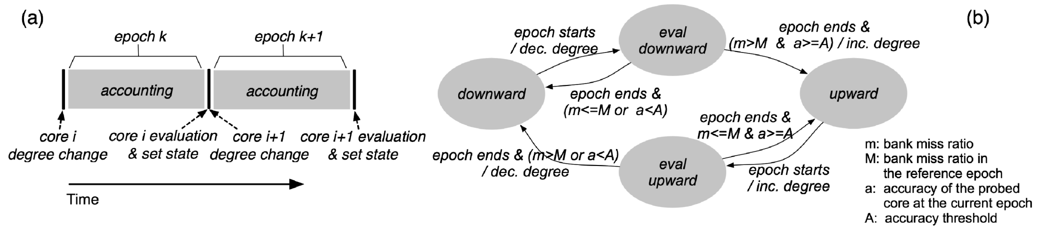

(a)
epoch k
epoch k+1
accounting
accounting
core i degree change
core i evaluation & set state
core i+1 degree change
core i+1 evaluation & set state
Time
(b)
eval downward
downward
upward
eval upward
epoch starts / dec. degree
epoch ends & (m>M & a>=A) / inc. degree
epoch ends & (m<=M or a<A)
epoch ends & m<=M & a>=A
epoch ends & (m>M or a<A) / dec. degree
epoch starts / inc. degree
m: bank miss ratio
M: bank miss ratio in the reference epoch
a: accuracy of the probed core at the current epoch
A: accuracy threshold

Fig. 3. ABS controller operations. (a) Core selection and temporal sampling. (b) Finite state machine controlling the per-core prefetch aggressiveness.

variables and only considers one threshold. A performance and complexity comparison between HPAC and ABS is presented in Section 7.4.

## 4. THE ABS CONTROLLER

An *ABS controller* is an adaptive mechanism that sets dynamically the aggressiveness associated to each core on the prefetcher installed in a bank of a banked shared LLC. Every LLC bank has an ABS controller commanding the prefetcher of that bank. Thus the ABS controller of an LLC bank is able to associate different levels of prefetch aggressiveness to each core, and conversely ABS controllers in different banks can associate to the same core different levels of prefetch aggressiveness.

ABS control relies on a hill-climbing approach for finding the minimum of a function (the miss ratio of a bank[1]) that we assume to be dependent on a set of variables namely, the prefetching aggressiveness of each core in the bank. Time is divided into regular intervals called *epochs*. In each epoch, in order to establish a cause-effect relationship between change in aggressiveness and change in performance, the aggressiveness of only one core (the probed core) is varied. The point is that at each epoch, the observed change in the bank miss ratio is only due to a single aggressiveness change. At the end of the epoch an aggressiveness value is established for the currently probed core and this value remains unchanged until it is probed again. Furthermore, ABS controllers force the prefetch aggressiveness associated to a core to be decreased if its accuracy falls under a given threshold. The operation of ABS controllers involves two aspects: (i) selection of the core to probe and temporal sampling, and (ii) adaptive per-core aggressiveness control.

*Core Selection and Temporal Sampling*. At the beginning of each epoch a core is chosen in a round-robin fashion[2] and its current prefetch aggressiveness is changed. Then, at the end of the epoch, the effect of the change is evaluated by comparing the bank miss ratios observed during the current epoch and a reference epoch (Figure 3(a)). The change is undone if the current bank miss ratio is greater than the reference one. Otherwise, the change is confirmed and the current epoch is set as the new reference. So, the core selection and temporal sampling guarantee that there is always only one prefetch aggressiveness change between reference and current epochs. That change corresponds to the probed core at each epoch.

At the end of an epoch, an aggressiveness value is established for the currently probed core and this remains unchanged until the core is probed again. Note that if an application remains in a stable phase the ABS controller reaches a steady state only

[1]Ratio of bank demand misses to demand requests coming from all cores. Other performance indexes were also tested as the target function, such as global miss ratio, MPKI, or IPC. Although results were similar, these other indexes were discarded because they are more expensive to compute in terms of communication and hardware cost.

[2]A random order was also tested achieving slightly worse results.

broken by the glitches involved in testing sub-optimal configurations. Hill-climbing processes usually deals with functions that are not time-dependent. Thus, the process stops when no change can be found to improve the value reached. However, we know miss ratio is time-dependent because applications change their behavior over time. This has two important implications for the design of ABS. (i) Our algorithm never stops. The combination of aggressiveness able to minimize the miss ratio changes over time and ABS continually seeks that combination. (ii) When the miss ratio reaches the global minimum in the corresponding program phase, ABS will set the current epoch as the reference epoch and the current miss ratio as the rate to beat. So, as a lower miss ratio will no longer appear, ABS will never change the control actions, and worse, a similar behavior may occur during long program phases after reaching a local minimum. In order to remedy this situation, the number of epochs elapsed without updating the reference epoch is counted. When this count is equal to the number of cores, the mechanism sets the last epoch as the new reference. Updating the reference epoch in this way ensures that a new value is taken after probing all cores without experiencing a miss ratio decrease. We use epochs of 32K cycles. Other durations were tested without significant variation. Epochs based on counting a given number of events, like cache misses, were also tested without significant variations.

*Adaptive Per-Core Miss-Gradient Aggressiveness Control.* In an ABS controller, every core has a state which consists of a prefetch aggressiveness *degree* and a prefetch aggressiveness *trend* (downward or upward). At the beginning of the epoch in which a core is being probed, the state changes to eval-downward or eval-upward (Figure 3(b). Note that to probe a core, ABS only changes the aggressiveness following the trend associated with the core, instead of testing both possibilities (downward and upward) as they would do other implementations of hill-climbing.

Four events are locally counted in each LLC bank during each epoch, namely: (1) bank accesses from all cores, (2) misses from all cores, (3) prefetches issued by the core being probed, and (4) hits from the core being probed on prefetched lines. Sequential tagged prefetch uses a bit per cache line to tag the prefetched lines. This bit is set when a line is loaded in the LLC by a prefetch, and it is reset when the line is used for the first time. We use this bit to count the hits of the probed core on prefetched lines. At the end of an epoch two ratios are computed: bank miss ratio (bank misses/bank accesses) and prefetch accuracy for the core being probed (hits from the core in prefetched lines/prefetches from the core). Then the computed bank miss ratio is compared with the bank miss ratio of the reference epoch. If it has increased, the state changes to the reverse trend (from eval-downward to upward or from eval-upward to downward). Otherwise, the state changes to the initial trend and the reference bank miss ratio is updated. Accuracy is involved in the transitions that leave the eval-x states. It is required that the probed core has an accuracy higher than a threshold to go to the upward state. The rationale of this requirement is to avoid the increase in the aggressiveness of one core whose prefetches are almost useless. We have observed that the optimal threshold depends on the intrinsic accuracy of the prefetch engine. Aggressive prefetch engines such as sequential tagged need a higher threshold (more restrictive) than other more conservative prefetch engines like sequential streams. We use an accuracy threshold of 0.6 when controlling sequential tagged with variable degree and of 0.3 when controlling sequential streams.

## 4.1. Example

Figure 4 shows an example of how an ABS controller works in an LLC bank of a system with four cores. The degree scale (0, 1, 4, 8, 16) represents the prefetch aggressiveness. Time moves from left to right and is divided into fixed length intervals as shown in

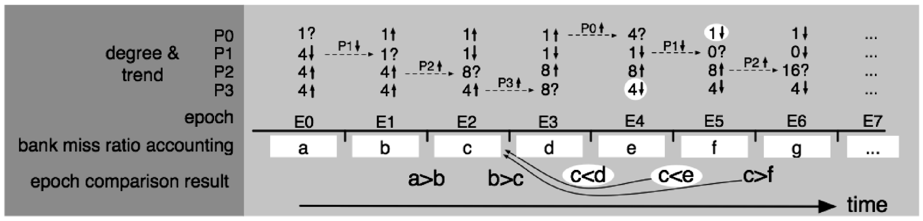


Fig. 4. Example of ABS controller working in an LLC bank of a 4-core system.

the *Epoch* row. The four rows designated as *degree & trend* (P0, P1, P2 and P3) show the level of prefetch degree and trend associated with each core at each epoch. For instance, 4↑ means prefetch degree of 4 and upward trend. The core identifier and its trend over the dashed arrows indicate the change applied between two consecutive epochs. For example, from E0 to E1 the degree of P1 is changed from 4 to 1. The *bank miss ratio accounting* row shows the miss ratio at each epoch. The last row shows the comparison result between the reference and the current bank miss ratios. Next we show the positive and the negative cases.

*Positive Aggressiveness Change.* At the beginning of E1, aggressiveness of the P1 core is changed from degree 4 to 1 following its downward trend. The question mark next to the degree of P1 at E1 epoch (1?) means that the change is being evaluated. At the end of E1 we observe a decrease in the bank miss ratio (a>b) with respect to the reference epoch (a). Therefore, the change in degree and the current trend of P1 are confirmed. E1 becomes the new reference epoch. The same happens at the epochs E2 and E5.

*Negative Aggressiveness Change.* White circles surround negative evaluations. At the beginning of E3, the P3 degree is changed from 4 to 8 following its upward trend. At the end of E3, an increase in the bank miss ratio (c<d) is observed, therefore the change in the P3 degree is undone, becoming 4, and its trend is reversed to downward. The reference epoch is not changed. At the beginning of E4, the P0 degree is changed from 1 to 4. Note that E2 is the reference epoch in E4. They only differ in the degree of P0, which is the one under evaluation in E4. At the end of E4 the bank miss ratio is also higher than it was at E2. Therefore, the change in the P0 degree is undone, the trend is reversed, and E2 remains as the reference in E5.

## 4.2. Miss Ratio as a Good Metric to Guide Aggressiveness

Prefetch related metrics such as coverage, accuracy, timeliness, pollution or consumed bandwidth have often been proposed to evaluate the quality of a prefetch engine because an aggregate figure such as the miss ratio does not allow the net effect of individual prefetches to be distinguished [Palacharla and Kessler 1994; Wallin and Hagersten 2003].

These same metrics were subsequently used in other works to guide prefetch aggressiveness [Dahlgren et al. 1993; Ebrahimi et al. 2009; Srinath et al. 2007]. However, these metrics are not directly related with system performance. Moreover, in the context of a banked LLC they pose two important problems: (i) some of them are hard to compute online, and (ii) they are difficult to aggregate in a single number in order to take a decision.

In this paper we use the LLC bank miss ratio as the main metric to guide prefetch aggressiveness. The penalty of the off-chip misses is large in processor cycles and so a miss ratio decrease has a great potential to reduce Cycles per Instruction (CPI) and improve performance. Therefore, we expect the LLC miss ratio to be a good measure of performance. Moreover, the ABS controller associated with each LLC bank can locally count the number of misses in the bank. In consequence, our proposal establishes the feedback loop without requiring communication among LLC banks.

From a performance standpoint, the prefetch related metrics are highly correlated with the miss ratio. In fact, some of these metrics are only valuable when they really correlate with the miss ratio. For instance, a prefetched line is considered useful if it is used along its lifetime in a cache, and useless otherwise (accuracy metric). However, a useful prefetch does not always have a positive impact on performance. Indeed, it will only increase performance if it gets a reduction in the miss ratio. In particular, if the line evicted by the prefetch is referenced before the prefetched line itself, despite having a useful prefetch, the miss ratio does not change and therefore no performance increase will be seen.

### 4.3. ABS Controller Hardware Cost

The hardware cost of our proposal is low. In each bank, an ABS controller needs 4 bits per core in order to keep its prefetch state (1 bit for trend + 3 bits for aggressiveness level). It also needs four 16-bit counters (bank misses, bank accesses, prefetch requests, and hits on prefetched blocks). Additionally, it needs a 3-bit counter (4 bits in a 16-core system) to maintain the reference epoch age (number of epochs without changing the reference). The reference and the current miss ratios are stored in 16-bit registers. Finally, a 15-bit counter is needed to divide time into 32K-cycle epochs. For instance, in an 8-core multiprocessor each ABS controller needs 146 bits. Thus, in our baseline system fitted with 4 LLC banks, 584 bits are needed. If we consider a 16-core system with the same memory hierarchy, 716 bits are needed by the four ABS controllers.

Most prefetchers add to every cache line a tag bit in order to detect first use after prefetch and then react based on the prefetchless miss stream[3] [Nesbit and Smith 2005; Smith 1982]. For instance, sequential tagged prefetch uses the tag bit to trigger new prefetches on the first use of a prefetched line. The ABS controller uses the same tag bit to count hits on prefetched lines, and so we ignore that bit in the ABS costs.

## 5. PREFETCHING FRAMEWORK

### 5.1. Prefetch Engine

The operation of the ABS controller is orthogonal to the prefetch engine used to generate prefetch requests. The only necessary characteristic in this prefetch engine is that it has to be able to operate at different aggressiveness levels. In this work, the base system uses a sequential tagged prefetcher with variable degree (degree varying along the following scale 0, 1, 4, 8, 16). Given an initial address and a degree $k$, it is asked to generate $k$ sequential references in the next $k$ cycles. In Section 7.4 we also evaluate ABS controlling sequential streams as the prefetch engine, using the same model of sequential streams as Ebrahimi et al. [2009]. ABS can control prefetch engines generating non-consecutive references, either belonging to a fixed-stride stream, or a stream generated by a context predictor like GHB or PDFCM [Nesbit and Smith 2005; Ramos et al. 2011]. Considering such prefetch engines under ABS control is an open research avenue, but we consider it is outside the scope of this work.

### 5.2. Bank-Isolated Prefetch

A common mapping of memory lines to LLC banks is line-address interleaving. On a miss on line L mapped to bank B, a sequential prefetcher located at bank B generates

[3]Given two equal caches, with and without prefetch, note that the miss stream outgoing from the prefetch cache differs from that of the prefetchless cache. However, it is possible to rebuild the prefetchless miss stream in a prefetch cache by joining the actual miss stream with the first-use stream of prefetched blocks.

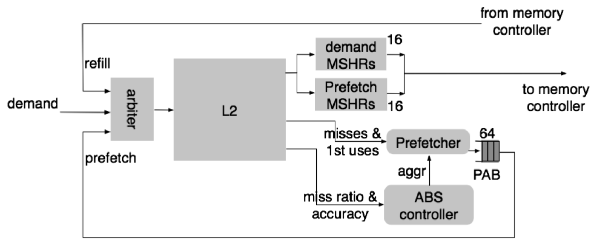


Fig. 5. Components of an LLC bank.

a prefetch of line L+1, which maps to the next LLC bank. The address is looked up in the destination cache bank and, on a miss, it is forwarded to the main memory. Thus, communication among LLC banks is required in order to send every prefetch request from the bank that generates it to the destination bank. Alternatively, we could send the prefetch request to the memory without a previous lookup. In this case, we can waste memory bandwidth on prefetching lines already existing in the LLC.

In order to avoid expensive communication between LLC banks or waste of memory bandwidth, LLC prefetch is arranged to achieve isolation among banks, i.e. prefetches generated from a bank always target itself. We analyze two bank-isolated prefetch methods: increasing stride and changing the address interleaving among banks. The first consists on increasing the prefetch stride from one to a multiple of the number of banks. As an example, in an LLC with 4 banks and a sequential prefetcher at each bank, a miss on the line L issues the prefetch of the line L+4. Achieving bank-isolation in this way has the drawback that several prefetchers have to learn a fraction of the same stream, and thus the number of prefetch addresses not issued during the learning time is multiplied by the number of banks, this is a serious drawback, especially for short streams.

The second method consists of increasing the address interleaving granularity among banks. The LLC banks are interleaved using operating system pages, where consecutive physical pages map into consecutive LLC banks (mapping logical to physical addresses is performed in our experiments by the simulated operating system). This way an address stream always maps to a single bank while the page boundary is not crossed, and all addresses generated by a bank prefetcher will target the same bank. This is not a problem because in order to avoid translating prefetch addresses, prefetchers do not usually issue addresses beyond the page boundary of the address originating the prefetch [Le et al. 2007; Hennessy and Patterson 2007; Intel 2011].

We have evaluated the performance of both interleaving options for the baseline system without prefetch, noting that performance was slightly higher using page interleaving. Similar conclusion was obtained in a previous work [Cho and Jin 2006].

## 5.3. Prefetch Details

Aggressive prefetchers such as a high-degree sequential tagged prefetcher can generate a significant number of prefetches. We assume the hardware cost of the prefetcher is lowered by sharing the same lookup port for demand and prefetch requests (demand requests have higher priority). Furthermore, only one adder is provided to each LLC bank in order to generate prefetches, and so prefetch addresses are computed at a rate of one per cycle. The generation of a burst of prefetch requests after a cache miss or cache hit to a tagged line is cut off if another event initiates a new prefetch burst.

After being generated, each prefetch is sent to a prefetch address buffer (PAB) and waits in a queue until the LLC lookup port is available, see Figure 5. Because the PAB has a finite number of entries, it is managed in FIFO order (both for servicing and dropping prefetch requests the oldest one is processed first). Before inserting a prefetch address, the PAB is checked for an already allocated entry with the same address in order to avoid having duplicated requests.

Prefetch and demand Miss Status Holding Registers (MSHRs) keep the requests to the main memory until the arrival of the corresponding cache lines. There are no duplicated entries between MSHRs. When the LLC bank tag port is available, the request at the head of the PAB looks up both the bank tags and the two MSHRs. Only if they all miss, is the request sent to the memory and inserted in the prefetch MSHRs.

Demands have higher priority than prefetch requests at every arbiter in the hierarchy. Moreover, a demand can arrive at an LLC bank asking for a line that is being prefetched but whose data are not yet loaded into the cache. In that case a prefetch upgrade command is sent to the memory controller. If the request is queued at the controller. That command will upgrade the prefetch request giving it demand priority. The rationale of this mechanism is to prevent an aggressive prefetch from damaging regular memory instructions.

## 6. METHODOLOGY

### 6.1. Experimental Setup

*Simics*, a full-system execution-driven simulator, has been used to evaluate our proposal [Magnusson et al. 2002]. The *Ruby* plugin from the *Multifacet GEMS* toolset was used to model the memory hierarchy with a high degree of detail [Martin et al. 2005], including coherence protocol, on-chip network, communication buffering, contention, etc. The prefetch system has been integrated into the coherence mechanism and a detailed DDR3 DRAM model has been added.

Multiprogrammed *SPECCPU 2K6* workloads running on a Solaris 10 Operating System have been used. In order to discard the less demanding memory applications and locate the end of the initialization phase, all the SPARC binaries were run on a real machine until completion with the reference inputs and hardware counters were used. Eight applications were subsequently discarded and 21 selected, shown in the first column of Table I.

For an eight-core system, a set of 30 random mixes of 8 programs each, were taken from the previously selected 21 *SPECCPU 2K6* programs (no effort has been made to distinguish between integer and floating point). In the next section, we usually show averages over this set of 30 mixes, but in order to gain a deeper insight into individual mix behaviors, sometimes a subset of 10 mixes is shown. This randomly chosen subset of mixes appears in Table I, along with the misses per kilo-instruction (MPKI) of each application in each mix when the application runs alone, that is, it runs using all the shared memory resources and with the remaining seven cores stopped.[4]

Summarizing, we have created 30 checkpoints of 8 applications each. Each checkpoint guarantees that no application is in its initialization phase. The cycle-accurate simulation starts at these checkpoints, warming the memory hierarchy for 500 million cycles with the prefetch disabled. We then collect statistics for the next billion cycles.

### 6.2. Baseline System

The baseline system has eight in-order cores. The shared LLC has four banks interleaved at page granularity (4KB in the simulated operating system). A MOSI protocol

[4]Notice that in general the same application appearing in two different mixes does not have the same MPKI value. This is because the number of executed instructions before creating a multiprogrammed checkpoint is given by the application with the longest initialization phase in the mix. This means that the results for a particular application, in general, can not be compared between mixes.

Table I. MPKI of the Benchmarks in the Selected Mixes

| | mix1 | mix2 | mix3 | mix4 | mix5 | mix6 | mix7 | mix8 | mix9 | mix10 |
|---|---|---|---|---|---|---|---|---|---|---|
| bzip2 | | | | | 1.8 | | 1.7 | | | |
| bwaves | 20.7 | | 20.7 | | 20.7 | 20.7 | 20.7 | | | 20.7 |
| mcf | 33.9 | | | 37.5 | 30.7 | 33.2 | 20.2 | | | |
| milc | 16.4 | 16.2 | | 34.2 | | | | | | |
| zeusmp | 16.7 | | | | | | 8.1 | 9.1 | 7.3 | 13.8 |
| gromacs | | | 3.9 | | | | | | 4.0 | |
| cactusADM | | 4.2 | | 4.1 | | 4.2 | 4.2 | 4.4 | | |
| leslie3d | | | | 28.3 | 32.5 | 36.4 | | | 14.4 | |
| gobmk | | | 1.5 | 1.5 | | | | | 1.4 | 1.5 |
| dealII | | 0.0 | | 0.1 | | | | 0.2 | 0.3 | |
| soplex | 3.0 | | | 4.0 | | | | 3.6 | | |
| povray | | 0.3 | 0.3 | | | | 0.3 | | 0.3 | 0.3 |
| calculix | | | | | | 0.5 | | | 5.9 | 0.5 |
| gemsFDTD | 32.5 | | 26.9 | | | | 32.5 | 26.8 | | |
| libquantum | | | | | 28.8 | 85.5 | 65.6 | | | |
| tonto | | 3.4 | 1.5 | | | | | | 1.6 | |
| lbm | 36.1 | 47.6 | 36.1 | | 36.1 | | | | | 36.1 |
| omnetpp | 0.7 | 5.4 | | | 0.7 | 0.7 | | 0.6 | | 0.7 |
| wrf | | | | | 3.0 | | | 0.5 | | 0.5 |
| sphinx3 | | 12.5 | | 12.9 | | | | | | |
| xalancbmk | | | 1.8 | | | 1.8 | | 1.5 | | |
| mix MPKI | 16.9 | 4.62 | 5.05 | 7.67 | 10.8 | 10.37 | 9.06 | 3.22 | 8.03 | 4.04 |

Table II. Baseline System Configuration

| | |
|---|---|
| Private L1 I/D | 16KB, 4-way pseudo LRU replacement, 64B line size, 1cycle |
| Shared L2 | 4MB inclusive (4 banks of 1MB each. Data array internally sub-banked), 4KB interleaving, 64B line size. Each bank: 16-way pseudo LRU replacement, 2-cycle TAG access, 4-cycle data access. 16 demand + 16 prefetch MSHR |
| Prefetch engine | Sequential tagged, degree 16 |
| DRAM | 1 rank, 16 banks, 4KB page size, Double Data Rate (DDR3 1333Mhz) |
| DRAM bus | 667Mhz, 8B wide bus, 4 DRAM cycles/line, 16 processor cycles/line |

keeps the memory system coherent while allowing thread migration among cores.[5] A crossbar communicates the first level caches and the shared LLC banks. There is a single DDR3 memory channel. The DRAM memory bus runs at a quarter of the core frequency. Table II gives additional implementation details.

## 6.3. Performance Indexes

As previous work has shown, sequential prefetch delivers good results when only one program is executed in the system [Smith 1982; Ramos et al. 2011]. However, as pointed out in Section 2, this assumption is no longer true for a multiprocessor system because resources are shared among cores that interfere each other. Thus, the goal of the ABS controller is to control the prefetch aggressiveness on a per-core basis in such a way that programs running together in the multicore system attain similar performance as when running alone with an aggressive sequential prefetch. In consequence, our performance indexes use as references the IPC of the programs running alone on a system with a sequential tagged prefetcher with a fixed degree of 16.

In order to evaluate the ABS controllers in the multiprocessor system, we use two system-oriented performance indexes, namely weighted speedup (*WS*) (Eq. (1)) [Luo et al. 2001], and main memory bandwidth consumption, and two user-oriented performance indexes, namely harmonic mean of speedups (*HS*) (Eq. (1)) [Snavely and Tullsen 2000], and fairness (*FA*) (Eq. (1)) [Mutlu and Moscibroda 2007]. The weighted speedup

[5]Migration can mainly affect to the operating system threads. Each application thread is bound to a different core for the multiprogrammed workloads.

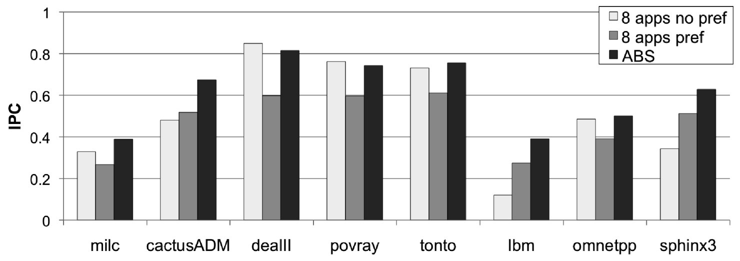


Fig. 6. IPC for eight SPEC2K6 applications (mix2) running on an 8-core system with a shared LLC.

quantifies the number of jobs completed per unit of time [Eyerman and Eeckhout 2008]. The harmonic mean of speedups is the inverse of the average normalized turnaround time [Eyerman and Eeckhout 2008]. To determine whether the co-execution in the multicore system benefits or harms some programs more than others we use the fairness index.

$$WS = \sum_{i=1}^{n} \frac{IPC_i^{MP}}{IPC_i^{SP}}, HS = \frac{n}{\sum_{i=1}^{n} \frac{IPC_i^{SP}}{IPC_i^{MP}}}, FA = \frac{min(IS_1, IS_2, \ldots, IS_n)}{max(IS_1, IS_2, \ldots, IS_n)}, where\ IS_i = \frac{IPC_i^{MP}}{IPC_i^{SP}} \quad (1)$$

$IPC_i^{SP}$: IPC of program i running alone in the system and with fixed degree-16 sequential tagged prefetch.
$IPC_i^{MP}$: IPC of program i when other applications run in the rest of the cores

## 7. RESULTS

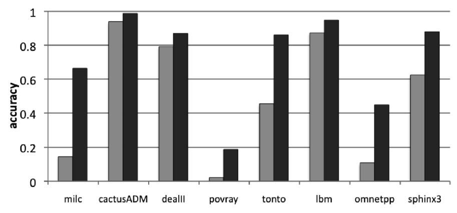
In this section we evaluate the ABS controllers. Section 7.1 analyzes ABS controlling sequential tagged prefetch in the baseline system. Section 7.2 shows results for 16-core systems. In Section 7.3 we increase the LLC size. In Section 7.4 ABS controllers are compared with a previous proposal, and in Section 7.5 they are evaluated using parallel applications.

### 7.1. Results for the 8-Core Baseline System

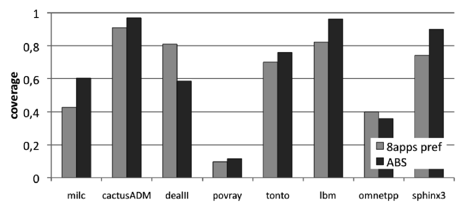
Figure 6 shows the results of ABS controlling prefetch aggressiveness of the *mix2* described in Section 2. The 8 programs run simultaneously and the bars corresponding to execution without prefetch (*8apps no pref*) and with fixed-degree (16) aggressive prefetch (*8apps pref*) are kept. A new bar corresponding to ABS prefetch (ABS) is added.

ABS controlled prefetch increases performance compared with the aggressive fixed degree prefetch in all programs. IPC improvement ranges from 23% in *sphinx3* to 40% in milc. With regard to the system without prefetch, ABS control only slightly affects the performance of *deall* and *povray*, while the aggressive prefetch leads to significant losses in five of the eight programs. In contrast, in six of the eight programs ABS control outperforms the system without prefetch, achieving improvements between 3% in *omnetpp* and 225% in *lbm*.

Global measures such as IPC do not allow us to find out how prefetching is working. To give insight into prefetching behaviour, Figure 7(a) and Figure 7(b) show prefetch coverage and accuracy, respectively, comparing fixed degree (*8apps pref*) with ABS-controlled degree (*ABS*). Prefetch coverage is very uneven across applications, ranging from about 0.1 in *povray* to more than 0.9 in *cactus*. ABS, despite being less aggressive, gets coverage similar or even higher than the fixed 16-degree prefetcher. ABS coverage is clearly better in five of the eight applications, and is clearly worse in two. As for accuracy, it is also uneven across applications, being very close to zero in *povray* and



(a) Prefetch accuracy



(b) Prefetch coverage

Fig. 7. Prefetch accuracy and coverage of sequential tagged prefetching with fixed degree of 16, and variable ABS-controlled degree (mix2).

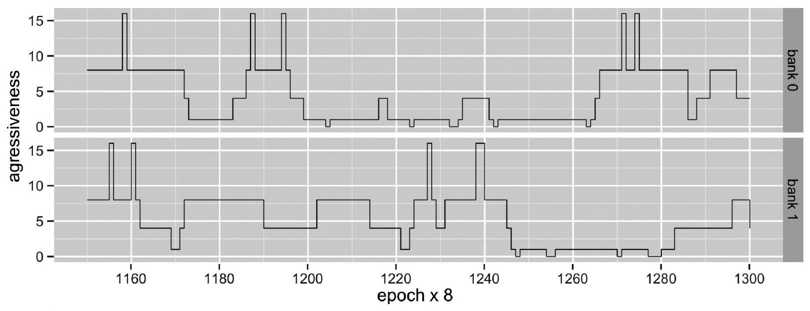


Fig. 8. Evolution of the prefetching aggressiveness level for the *milc* application (mix2) in two of the four LLC banks during 160 tests.

very close to one in *cactus*. But here ABS is the clear winner, achieving higher accuracy in all applications, highlighting the cases of *milc*, *povray*, *tonto*, *omnetpp* and *sphinx3*. The combination of both metrics, similar coverage and better accuracy, explains the performance improvement obtained by ABS.

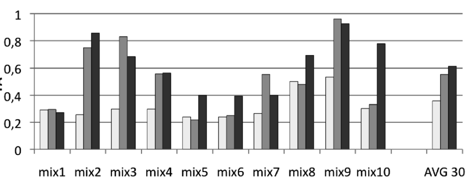
It is interesting to delve into how different the aggressiveness computed in each bank by the replicated ABS controllers can be. Figure 8 plots a temporal trace of the prefetch degree for application *milc* in the mix2, in a prefetch system under ABS control in two of the four LLC banks (*bank 0* and *bank 1*). The failed tests (glitches), accounting for 1/ 8 of the total time in the worst case, have been removed to smooth the plot.

In the plotted sample, both ABS controllers usually take different control decisions; e.g. at time 1180, bank 0 prefetchs with degree 2, while bank 1 uses degree 8. The plot shows the flexibility of the distributed ABS control: a particular core may issue a miss stream with a different pattern into each LLC bank.

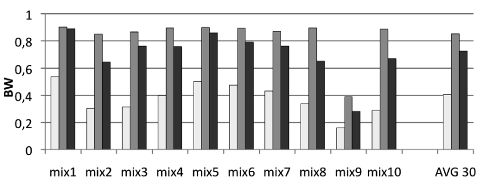
Figure 9 shows HS, WS, FA and consumed bandwidth for systems without prefetch (*no pref*), with fixed-degree aggressive prefetch (*aggr pref*), and under ABS control (*ABS*) for the ten mixes shown in Table I. In each plot the rightmost bar group is the average of the 30 mixes (*AVG30*).

The HS values show a nonuniform pattern across the different mixes (Figure 9(c)). Aggressive prefetch increases by 4% the average HS with respect to no prefetch, but causes losses in six of the 10 mixes. Under ABS control, prefetch improves in 9 of the 10 mixes (up to 60% in mix6) and produces small losses in the other mix (0.5% in mix9). Also, ABS control always increases performance compared to no prefetch, between 20% and 50% in mix3 and mix6, respectively. On average, prefetch under ABS control, improves the system without prefetch by 35%.

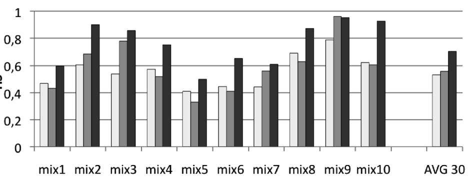
In terms of WS (Figure 9(d)), aggressive prefetch causes losses in seven of the 10 mixes and performs on average 3% worse than the system without prefetch. ABS control improves aggressive prefetch in 9 of the 10 mixes (up to 47% in mix4) and produces



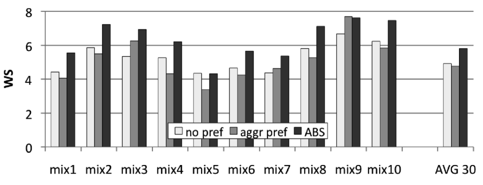
(a) Fairness



(b) Bandwidth consumed



(c) Harmonic mean of speedups



(d) Weighted speedup

Fig. 9. Results for ten mixes of SPEC2K6 applications running on an 8-core system with a shared LLC.

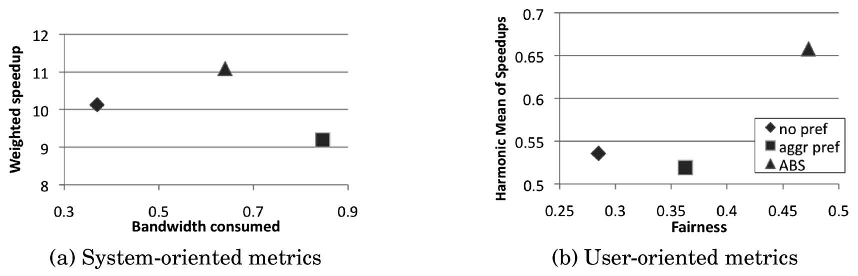
negligible losses in the other mix (1% in mix9). On the other hand, prefetch under ABS control improves the system without prefetch in 9 of the 10 mixes, with improvements ranging from 14% in mix9 to 27% in mix3. In mix5, WS results in a reduction of 0.3%. On average, prefetch controlled by ABS improves the system without prefetch by 18%.[6]

Figure 9(a) plots the FA values, showing that the system with aggressive prefetch is significantly more fair than the system without prefetch. This is because the performance indexes use as reference a system with prefetch as we have seen in Section 6.3. Therefore, in the system without prefetch we see the unfairness introduced by the lack of prefetch itself, plus the unfairness due to the interferences among the eight cores. In Figure 9(a), we observe the low fairness of mix2 in the system without prefetch. As the HS index includes some notion of fairness in its definition and WS is a pure throughput index, the previous issue about mix2 becomes clear. On average, the system using ABS controllers is more fair than the system with aggressive prefetch (0.62 and 0.56, respectively). ABS controllers make the system more fair in 6 of the 10 mixes (differences between 1% and 140%), and less fair in the remaining 4 (differences between −3% and −30%).

Finally, in Figure 9(b) we see that the main memory bandwidth consumption of the system without prefetch is very uneven among the different mixes, varying between 18% and 55% of the maximum bandwidth. However, the common pattern is that aggressive prefetch greatly increases bandwidth consumption with respect to the system without prefetch (on average, from 40% to 85% of maximum bandwidth), and ABS removes a significant portion of that increase lowering it to 70% of maximum bandwidth.

Summarizing, in an 8-core chip with a shared 4-MB LLC, the use of ABS controllers improves the system that is using uncontrolled aggressive prefetch. On average, throughput (*WS*), the inverse of the turnaround time (*HS*), and fairness (*FA*) increase 27%, 23% and 11%, respectively while memory bandwidth consumption decreases by 18%.

[6]Notice that the performance indexes HS and WS do not always correlate; in mix2, for instance, the HS index indicates that aggressive prefetch is better than no prefetch while the WS index indicates the contrary. The discussion on fairness will give a deeper insight into what is happening.



(a) System-oriented metrics (b) User-oriented metrics

Fig. 10. ABS performance on a 16-core system.

### 7.2. Results for a 16-Core System

In this section we analyze the behavior of prefetch in a 16-core system. The LLC is not modified, so that the increase in the number of cores results in an increased pressure on the LLC. However, communication with the main memory is expanded from one to two DDR3 channels. We run 30 mixes of 16 applications randomly selected among the 21 SPEC CPU 2006 shown in Table I. We only present the average of each index over the 30 mixes of 16 programs each. Figure 9(a) combines in a single Y-X plot the system-oriented metrics, WS and bandwidth, while Figure 9(b) combines the user-oriented metrics, HS and FA.

In a system with 16 processors, fixed-degree (16) aggressive prefetch (*aggr pref*) produces losses compared to no prefetch (*no pref*) in terms of throughput (WS decreases 9%) and turnaround time (HS decreases 4%). The memory bandwidth consumption greatly increases from 36% to 85%. Only fairness improves from 0.28 without prefetch to 0.36 with aggressive prefetch.

Controlling aggressiveness leads to improvements in all metrics. Compared to aggressive prefetch, ABS control (*ABS*) increases the HS index by 27% (22% compared to no prefetch), increases the FA index to 0.48, and also improves the system throughput index with a WS increase of 25% (14% compared to no prefetch). The bandwidth consumption decreases significantly compared to aggressive prefetch, from 85% to 62% of the maximum, but it is still greater than without prefetch which only requires 36% of the maximum.

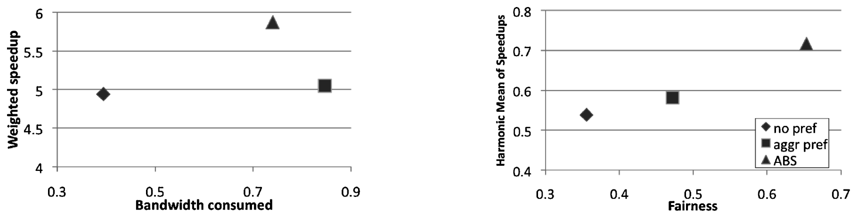
Summarizing, in a 16-core chip with a shared 4-MB LLC, ABS improves the system in all indexes. Comparing between 16 and 8 cores, the increase in the WS index is similar but the improvement in the rest of the indexes, HS, fairness and memory bandwidth, is much higher. This result is consistent because the pressure on the memory hierarchy in a 16-core chip is larger than in an 8-core, resources are more scarce, and therefore controlling the prefetch aggressiveness becomes more important.

### 7.3. Doubling the LLC Size

In this section we analyze the behavior of prefetch in the 8-core baseline system when doubling the LLC size to 8 MB. We only show average indexes computed over the 30 mixes already used in section 7.1. The system-oriented metrics WS and bandwidth, are shown in Figure 10(a), while Figure 10(b) shows the user-oriented metrics HS and Fairness.

In an 8-core chip with a shared 8-MB LLC, the use of ABS controllers also improves the behavior of uncontrolled aggressive prefetch. On average, throughput (*WS*), the inverse of the turnaround time (*HS*), and fairness (*FA*) increase 18%, 24% and 38%, respectively while memory bandwidth consumption decreases 14%.

When increasing the cache size, controlling the prefetch aggressiveness becomes less important for improving performance, but it improves fairness and saves bandwidth.



(a) System-oriented metrics (b) User-oriented metrics

Fig. 11. ABS performance on an 8-core system with an 8MB LLC.

Thus, when increasing from 4 to 8 MB, ABS improvements over uncontrolled prefetch change from 27% to 18% in *WS*, from 11% to 38% in *FA*, and from 18% to 14% in *BW*. As for *HS*, the results are similar.

### 7.4. HPAC Comparison

Next we compare the ABS control with the *Hierarchical Prefetcher Aggressive Control mechanism (HPAC)* introduced in Ebrahimi et al. [2009]. To the best of our knowledge, this was the only work to date on adjusting prefetch aggressiveness in a shared LLC.

HPAC works in a centralized LLC with a single access port although internally it is organized in banks to support several concurrent accesses. The proposal uses sequential streams as the prefetch engine and a local control of aggressiveness for each core: Feedback-Directed Prefetching (FDP) [Srinath et al. 2007]. HPAC adds a global interference feedback in order to coordinate the prefetchers of the different cores and throttle their aggressiveness.

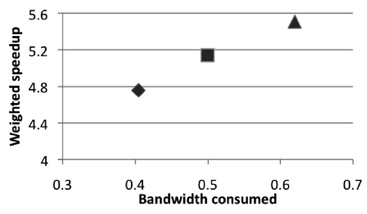
Since we assume autonomous LLC banks, possibly placed at distant die locations, distributing HPAC is not straightforward. We choose a distributed implementation giving HPAC as much knowledge and control as possible, namely each core has an FDP per LLC bank, and each LLC bank has an HPAC controlling the corresponding FDP. So, the distributed HPAC/FDP we test requires 32 FDPs (8 FDPs per bank × 4 banks = 32 FDPs), and 4 HPACs (1 HPAC per bank × 4 banks = 4 HPACs).

Besides other local bank metrics, HPACs gather statistics from one/two memory controllers (8/16 core systems), and the communication among HPACs and the memory controllers is modelled in an ideal way (zero-delay/no *BW* limitations).

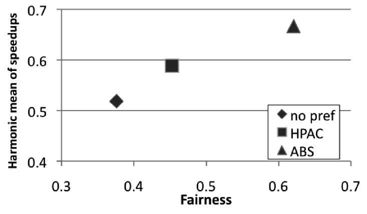
We have used the thresholds indicated in the published proposals for both mechanisms. We simulate 32 streams per core and LLC bank (32 streams × 8 cores × 4 banks = 1024 streams). Each stream launches sequential prefetches with a degree and distance from a starting address. We implement five levels of aggressiveness that correspond to degrees 1, 1, 2, 4, and 4 and distances 1, 4, 16, 32, and 64, respectively. The aggressiveness control mechanism (HPAC/FDP or ABS) decides the aggressiveness level associated to each core.

Figure 12 plots the results for HPAC and ABS on an 8-core system. Both mechanisms use sequential streams as the prefetch engine. Performance indexes have been computed using as references the IPCs of the programs running alone on a system with a sequential stream prefetcher with a fixed level of aggressiveness (distance 64 and degree 4). We only show average indexes computed over the 30 mixes already used in Section 7.1.

Figure 12(a) shows the system-oriented metrics, *WS* and *BW*, while Figure 12(b) shows the user-oriented metrics, *HS* and *Fairness*. ABS control obtains better results than HPAC in all metrics except in the consumed bandwidth. ABS improves *WS* index by 8%, *HS* index by 14% and fairness by 40%. Compared to no prefetch, bandwidth consumption is higher in ABS (62%) than in HPAC (50%).

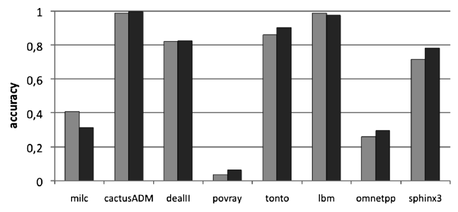


(a) System-oriented metrics

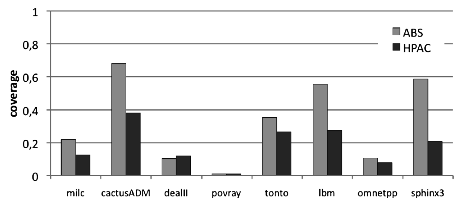


(b) User-oriented metrics

Fig. 12. HPAC and ABS performance on an 8-core system.



(a) Prefetch accuracy



(b) Prefetch coverage

Fig. 13. Prefetch accuracy and coverage of sequential streams with variable ABS-controlled and HPAC-controlled degrees (mix2).

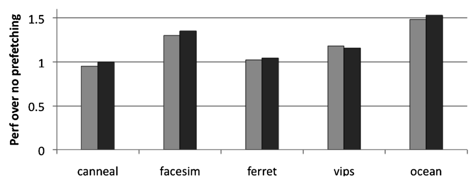
The results for a 16-core system are not shown but are similar. ABS also produces better results than HPAC in all metrics except consumed bandwidth. ABS improves the *WS* index by 7%, the *HS* index by 11%, and fairness by 29%. Compared to no prefetch, bandwidth consumption is higher in ABS (54%) than in HPAC (44%).

Figure 13(a) and Figure 13(b) show prefetch accuracy and coverage, respectively, of sequential streams prefetching, with ABS and HPAC controlling the prefetch aggressiveness. Sequential streams, as a prefetch engine, are less aggressive than sequential tagged, because the former has a long learning period and the latter has not. This explains the low coverage we see compared to that of Figure 13(b). On the other hand, ABS shows greater coverage than HPAC for most applications in the mix. This may be so because HPAC can be more restrictive than ABS. For instance, in a situation of low accuracy and high pollution, HPAC throttles prefetch aggressiveness regardless of its overall impact on the miss ratio. In contrast, ABS throttles prefetch only if the miss ratio grows. As for accuracy, Figure 13(a) shows that, despite being more aggressive, ABS gets accuracy similar to that of HPAC.

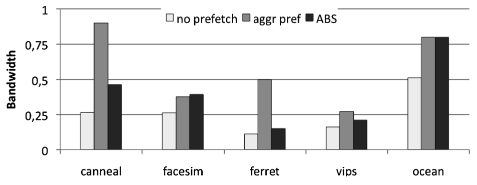
From the referenced papers we can compute accurately the HPAC/FDP hardware costs [Ebrahimi et al. 2009; Srinath et al. 2007]. In the 8-core system having 4 LLC banks of 1MB each, the total budget to implement HPAC/FDP is 466,624 bits (196,608 + 4 × (33,664 + 33,840)).[7] The implementation cost of the 1024 sequential streams, and the prefetched bit in each cache line have not been accounted for. In the 16-core system with the same LLC the FDP/HPAC cost is 933,056 bits.

Summarizing, the ABS controller gives a better performance than HPAC at the expense of some increase in consumed bandwidth. In addition, it succeeds with a very low implementation cost.

[7]FDP requires a core identifier per LLC block (3 bits × 64K blocks = 196,608 bits). In each bank, FDP also requires seven 16-bit counters per core, and a 1-Kentry Bloom filter per core to detect intra-core prefetching interferences (1 bit and a core id per entry, totaling 33,664 bits per bank). In turn, HPAC requires in each bank eight 16-bit counters per core, three more 16-bits counters, and a 1-Kentry Bloom filter per core (1 bit and a core id per entry) to detect inter-core prefetching interferences (33,840 bits per bank).



(a) Speedup over no prefetch



(b) Bandwidth consumed

Fig. 14. Results for five parallel applications.

## 7.5. Results with Parallel Applications

In this section we analyze the behaviour of our proposal when running parallel applications in the baseline system presented in Section 6.2. A priori, this behaviour should not depend on whether the threads running on each processor are independent or not. ABS varies the prefetch aggressiveness associated with each core seeking to optimize the performance of each LLC bank in terms of bank misses. The decrease in the number of misses in each bank should result in improving system performance.

We have selected the five applications of the *PARSEC* [Bienia 2011] and *SPLASH-2* [Woo et al. 1995] suites which have more than 1 MPKI in a 4-MB LLC (concretely canneal, facesim, ferret, vips and ocean applications. And their MPKIs are respectively 4.48, 3.45, 1.27, 1.74, and 13.35). In order to avoid migration, the threads are bound to cores using system calls. Binding is not performed if an application spawns in its parallel phase more threads than there are cores in the system. Performance statistics (execution time, memory bandwidth) are only taken in the parallel phases, which are run to completion in all the applications. We utilize the *simmedium* input set for PARSEC applications and a 1026x1026 grid for *Ocean*.

Figure 14(a) shows speedups of a system with fixed-degree prefetch (*aggr pref*) and with prefetch under ABS control (*ABS*) compared to a system without prefetch. The fixed-degree prefetch improves performance with respect to no prefetch in four out of the five applications. Only *canneal* suffers a 5% increase in execution time when under prefetch with fixed-degree. The ABS control achieves higher speedups than fixed-degree prefetch in all the applications except *vips* (−1.8%). In *canneal*, the ABS controllers reduce prefetching looses from 5% to 0.3%.

Figure 14(b) shows the memory bandwidth consumption of the system without prefetch (*no prefetch*), with fixed-degree prefetch (*aggr pref*), and with prefetch under ABS control (*ABS*). Fixed-degree prefetch significantly increases the memory bandwith consumption for all the applications (from 40% in *facesim* to 450% in *ferret*). The ABS control reduces memory bandwidth consumption with respect to fixed-degree in *canneal* (−50%), *ferret* (−70%), and *vips* (−23%), and slightly increases it in *facesim* (+5%). Summarizing, the ABS controllers also improve performance over fixed-degree prefetch when running parallel applications, in spite of the low miss ratios of these applications. Execution time is reduced in four out the five analyzed parallel applications. Besides, significant savings in memory bandwidth arise in three applications.

## 8. CONCLUSIONS

In this paper we focus on the shared, last level cache (LLC) of a multicore chip. We assume a LLC having a distributed implementation in multiple banks. In this scenario, we introduce the ABS controller (Adaptive prefetch control for a Banked Shared LLC), an adaptive mechanism to control the prefetch aggressiveness independently for each core in each LLC bank. The ABS controller implements a hill-climbing algorithm which runs stand-alone at each LLC bank, using only local information. Bank miss ratio and prefetch accuracy are sampled at fixed-length epochs and used as performance index.

For each bank at each epoch the aggressiveness of only one core is varied in order to establish a cause-effect relationship between the change in aggressiveness and the change in the performance index.

The ABS controllers are evaluated to adjust the aggressiveness level of variable-degree sequential tagged prefetchers. Our analysis using multiprogrammed SPEC2K6 workloads shows that the mechanism improves both user-oriented metrics (Harmonic Mean of Speedups by 27% and Fairness by 11%) and system-oriented metrics (Weighted Speedup increases by 22% and Memory Bandwidth Consumption decreases by 14%) over an eight-core baseline system that uses aggressive sequential prefetch with a fixed degree. Similar results have been obtained on a sixteen-core system, when doubling the LLC size or running parallel applications.

Besides, ABS is also compared to the mechanism for a centralized shared LLC proposed by Ebrahimi et al. [2009] but adapted for a banked LLC. For this comparison, variable-aggressiveness sequential stream prefetchers are used as prefetch engines. ABS performs better in all the performance indexes except in required bandwidth, which is somewhat greater.

Summarizing, ABS controllers are able to control the aggressiveness of prefetch engines in a distributed fashion and with very low implementation costs. Their distributed nature assures scalability in the number of cores and cache banks for future multicore chips.

## REFERENCES

Bienia, C. 2011. Benchmarking modern multiprocessors. Ph.D. thesis, Princeton University.

Cantin, J. F., Lipasti, M., and Smith, J. E. 2006. Stealth prefetching. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS-XII.

Cho, S. and Jin, L. 2006. Managing distributed, shared l2 caches through os-level page allocation. In *Proceedings of the 39th International Symposium on Microarchitecture*.

Conway, P., Kalyanasundharam, N., Donley, G., Lepak, K., and Hughes, B. 2010. Cache hierarchy and memory subsystem of the amd opteron processor. *IEEE Micro 30*, 16–29.

Dahlgren, F., Dubois, M., and Stenstrom, P. 1993. Fixed and adaptive sequential prefetching in shared memory multiprocessors. In *Proceedings of the 22nd International Conference on Parallel Processing*.

Ebrahimi, E., Mutlu, O., Lee, C. J., and Patt, Y. N. 2009. Coordinated control of multiple prefetchers in multi-core systems. In *Proceedings of the 42th Annual International Symposium on Microarchitecture*.

Eyerman, S. and Eeckhout, L. 2008. System-level performance metrics for multiprogram workloads. *IEEE Micro 28*, 42–53.

Hennessy, J. and Patterson, D. 2007. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann.

Intel. 2011. *Intel 64 and IA-32 Architectures Optimization Reference Manual*.

Kongetira, P., Aingaran, K., and Olukotun, K. 2005. Niagara: a 32-way multithreaded sparc processor. *IEEE Micro 25*, 21–29.

Koppelman, D. M. 2000. Neighborhood prefetching on multiprocessors using instruction history. In *Proceedings of the 9th International Conference on Parallel Architectures and Compilation Techniques*.

Kottapalli, S. and Baxter, J. 2009. Nehalem-ex cpu architecture. In *Hot Chips*.

Le, H. Q., Starke, W. J., Fields, J. S., O'Connell, F. P., Nguyen, D. Q., Ronchetti, B. J., Sauer, W. M., Schwarz, E. M., and Vaden, M. T. 2007. IBM power6 microarchitecture. *IBM J. Rese. Devel. 51*, 639–662.

Luo, K., Gummaraju, J., and Franklin, M. 2001. Balancing thoughput and fairness in smt processors. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software*.

Magnusson, P. S., Christensson, M., Eskilson, J., Forsgren, D., Hallberg, G., Hogberg, J., Larsson, F., Moestedt, A., and Werner, B. 2002. Simics: A full system simulation platform. *Computer 35*, 50–58.

Martin, M., Sorin, D. J., Beckmann, B. M., Marty, M., Xu, M., Alameldeen, A., K., M., Hill, M., and Wood, D. 2005. Multifacets general execution-driven multiprocessor simulator (gems) toolset. *SIGARCH Comput. Architect. News 33*, 2005.

Mutlu, O. and Moscibroda, T. 2007. Stall-time fair memory access scheduling for chip multiprocessors. In *Proceedings of the 40th International Symposium on Microarchitecture*.

NESBIT, K. J. AND SMITH, J. E. 2005. Data cache prefetching using a global history buffer. *IEEE Micro 25*, 90–97.

PALACHARLA, S. AND KESSLER, R. E. 1994. Evaluating stream buffers as a secondary cache replacement. In *Proceedings of the 21st International Symposium on Computer Architecture*.

RAMOS, L. M., BRIZ, J., IBÁÑEZ, P. E., AND VIÑALS, V. 2011. Multi-level adaptive prefetching based on performance gradient tracking. *J. Instruction-Level Paral. 13*, 1–14.

SMITH, A. J. 1982. Cache memories. *ACM Comput. Surv. 14*, 473–530.

SNAVELY, A. AND TULLSEN, D. M. 2000. Symbiotic jobscheduling for a simultaneous multithreaded processor. *SIGARCH Comput. Architec. News 28*, 234–244.

SOMOGYI, S., WENISCH, T. F., AILAMAKI, A., AND FALSAFI, B. 2009. Spatio-temporal memory streaming. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*.

SRINATH, S., MUTLU, O., KIM, H., AND PATT, Y. N. 2007. Feedback directed prefetching: Improving the performance and bandwidth-efficiency of hardware prefetchers. In *Proceedings of the 13rd International Symposium on High Performance Computer Architecture*.

TCHEUN, M., YOON, H., AND MAENG, S. R. 1997. An adaptive sequential prefetching scheme in shared-memory multiprocessors. In *Proceedings of the 26th International Conference on Parallel Processing*.

WALLIN, D. AND HAGERSTEN, E. 2003. Miss penalty reduction using bundled capacity prefetching in multiprocessors. In *Proceedings of the 17th International Parallel and Distributed Processing Symposium*.

WOO, S. C., OHARA, M., TORRIE, E., SINGH, J. P., AND GUPTA, A. 1995. The splash-2 programs: characterization and methodological considerations. In *Proceedings of the 22nd International Symposium on Computer Architecture*.

Received July 2011; revised October 2011; accepted November 2011