Perf-Insight: A Simple, Scalable Approach to Optimal Data Prefetching in Multicores

Muneeb Khan, David Black-Schaffer and Erik Hagersten
Department of Information Technology, Uppsala University
Email: {muneeb.khan, david.black-schaffer, eh}@it.uu.se

***Abstract*—Aggressive hardware prefetching is extremely beneficial for single-threaded performance but can lead to significant slowdowns in multicore processors due to oversubscription of off-chip bandwidth and shared cache capacity. This work addresses this problem by adjusting prefetching on a per-application basis to improve overall system performance. Unfortunately, an exhaustive search of all possible per-application prefetching combinations for a multicore workload is prohibitively expensive, even for small processors with only four cores.**

In this work we develop Perf-Insight, a simple, scalable mechanism for understanding and predicting the impact of any available hardware/software prefetching choices on applications' bandwidth consumption and performance. Our model considers the overall system bandwidth, the bandwidth sensitivity of each co-running application, and how applications' bandwidth usage and performance vary with prefetching choices. This allows us to profile individual applications and efficiently predict total system bandwidth and throughput. To make this practical, we develop a low-overhead profiling approach that scales linearly, rather than exponentially, with the number of cores, and allows us to profile applications while running in the mix.

With Perf-Insight we are able to achieve an average weighted speedup of 21% for 14 mixes of 4 applications on commodity hardware, with no mix experiencing a slowdown. This is significantly better than hardware prefetching, which only achieves an average speedup of 9%, with three mixes experiencing slowdowns. Perf-Insight delivers performance very close to the best possible prefetch settings (22%). Our approach is simple, low-overhead, applicable to any collection of prefetching options and performance metric, and suitable for dynamic runtime use on commodity multicore systems.

I. Introduction

Aggressive hardware prefetching has proven extremely successful at improving performance for single-threaded applications. Yet for mixed workloads on multicore processors such aggressive prefetching often hurts overall performance by inefficiently using shared memory system resources [7, 9, 10, 11, 19]. This penalty comes from both the inaccuracy of aggressive prefetching (fetching data that will not actually be needed) and the increase in data fetch rate, both of which consume shared off-chip bandwidth and evict other cores' data from the shared cache. The impact of these effects depends on each application's amenability to different prefetching approaches as well as their sensitivity to the system-wide bandwidth and cache pressure. As a result, the optimal set of prefetching choices depends on both the individual applications and the mix as a whole.

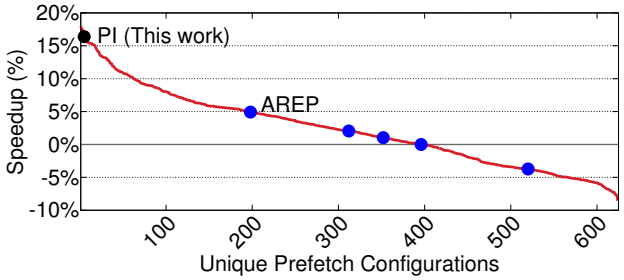


Figure 1. Distribution of measured performance gain (weighted speedup) over *No-Prefetching* across all 625 different possible prefetch combinations for a 4 application mix. The combinations are sorted by their overall performance. The 5 blue points highlight the settings where all applications have the same prefetch setting. In this mix the maximum speedup of previous work (AREP [11]) is 5%, while the maximum speedup can be up to 17% with the per-application prefetch settings from this work, Perf-Insight (PI).

To address this problem Khan, Laurenzano, Mars, Hagersten, and Black-Schaffer [11] proposed Adaptive Resource Efficient Prefetching (AREP), which dynamically explores prefetching strategies for mixes of applications to find the one with the best performance. While AREP was able to deliver an 8% throughput improvement over hardware prefetching by choosing among 5 combinations of hardware and software prefetching, it did so with the limitation of choosing the same strategy for all applications in the mix. The reason for this limitation was simply the exponential nature of the choices: for a 4-application mix with 5 different prefetching combinations there are $5^4 = 625$ possible combinations, and exploring this many possibilities at runtime was impractical. Further, while 625 possible combinations might be manageable with offline profiling, the exponential growth with the number of cores makes such an exhaustive search impractical in even the medium run: an 8-application mix presents a staggering 390,625 different prefetching combinations (Figure 2).

However, requiring the same prefetch strategy for each co-running application in a mix sacrifices a significant amount of performance, as each application may have different sensitivities to different prefetching strategies and to the

shared bandwidth and cache consumption of the other co-running applications. An example of this is shown in Figure 1, which plots the performance (weighted speedup) of all 625 combinations of 5 different prefetching strategies for a 4-application mix run on a commodity multicore (see Section III), sorted from fastest to slowest. The points where all applications have the same prefetching strategy are shown in blue, with the best (leftmost) of these labeled AREP. As can be seen from this data, allowing each application to have a different prefetching strategy has the potential to improve performance from AREP's best 5% gain to over 18%, as marked by the black dot on the far left. The challenge to obtaining this performance increase is in avoiding the need to explore the 625 combinations of prefetch strategies for the applications in each 4-application mix.

In this work we develop Perf-Insight, a simple method that allows us to efficiently model the impact of prefetch settings on each application in the mix and iteratively solve for the overall impact of each application in a mix. To develop the model, we first look at how applications respond to choices in prefetch strategy and the bandwidth pressure of other applications in the mix (Section III-A). From this analysis we conclude that we can simply and efficiently characterize applications in the mix by sampling a very small subset of all possible combinations (Section III-B). Critically, this subset grows linearly (Figure 2) with the number of cores, making it quite practical for use at runtime on multi-core machines.

We then use this method to develop an iterative solver to predict each application's bandwidth and performance within a mix (Section III-B), and then evaluate our approach with 14 mixes on a commodity multicore processor (Section IV). With Perf-Insight we are able to deliver performance that is significantly better than hardware prefetching alone and AREP, and nearly as good as oracle-based prefetching choices. Further, we show that Perf-Insight provides a robust means for choosing the best prefetching settings for a mix and describe how this can be readily applied at runtime.

This paper makes the following contributions:

- We present a quantitative analysis of the relationship between prefetch settings and applications' bandwidth and performance within mixes.
- Based on our analysis, we develop two simple models for characterizing the prefetch-dependent bandwidth and performance behavior of applications in a mix.
- Using the insights from our analysis we present a method to dramatically simplify the calibration of the two models, which allows us to scale linearly in the number of cores, rather than exponentially.
- We demonstrate that our model and simplified calibration approach allow us to accurately predict application performance and off-chip bandwidth for arbitrary prefetch combinations in the mix.
- We demonstrate how we can significantly improve overall performance of application mixes on commodity systems by choosing near-optimal per-application prefetch settings.

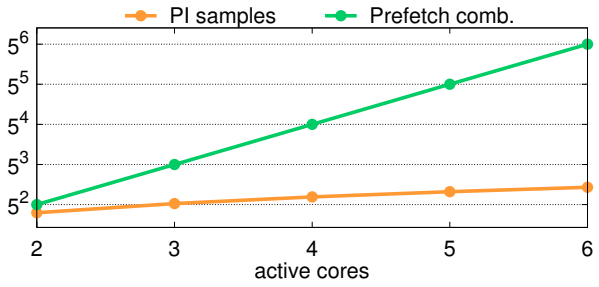


Figure 2. Scalability of Perf-Insight's model-based approach vs. the size of the prefetch options configuration space for per-application prefetch settings. The total number of prefetch combinations grows exponentially with each added core, while our approach grows linearly.

While this paper demonstrates the use of Perf-Insight for 5 specific prefetching options, it should be noted that the method itself is generic, and can work with any collection of available prefetching choices.

II. Background

A. Multicore Resource Sharing

There has been a significant amount of work on understanding the performance impact of shared resources in multicore processors. The Cache Pirate [4] and Bandwidth Bandit [5] techniques demonstrated how to quantitatively measure the impact of shared cache and bandwidth on application performance. To do so, they measured the performance of the target application while running in isolation with an artificial "pirate" application to steal shared resources. With this setup they could control precisely how much cache space and bandwidth was stolen and measure the impact of this sharing on the target application. This information allowed them to accurately predict the scaling of homogeneous application workloads based on the assumption that multiple copies of the same application will receive equal shares of the shared resources.

BubbleUp [18] demonstrated an alternative approach for qualitatively assessing the impact of shared cache and bandwidth on application performance. In BubbleUp, multiple "bubble" applications were co-run in isolation with the target application to stress different shared resources. From the impact of the bubbles on the target (and vice versa) they were able to deduce both the sensitivity of the target and the pressure it would exert on other applications. With this data they were able to efficiently *choose heterogeneous mixes* of applications to co-schedule with minimal performance degradation. Yang, Breslow, Mars, and Tang [26] present a similar yet dynamic online profiler and prediction mechanism to deliver satisfactory QoS.

In this work we develop a quantitative approach to modeling applications' sensitivities to off-chip bandwidth and

HWPF	Hardware Prefetching Only
HW+SWPF	Hardware and Software[1] Prefetching
L1HW+SWPF	L1 Hardware and Software Prefetching
SWPF	Software Prefetching Only
NOPREF	No Prefetching

The software prefetching technique is from [9], although this work is agnostic to the particular prefetching techniques available

Table I
THE 5 PREFETCHING SETTINGS EVALUATED IN THIS WORK.

prefetch settings and use this to improve performance for a *given* mix of heterogeneous applications. Our approach differs from previous work in that we can take our measurements as the applications run together in the mix and improve performance for a given mix by adjusting the hardware and software prefetch settings.

B. Prefetching Strategies

The Perf-Insight approach developed in this work is agnostic to the particular prefetching strategies available as long as they can be selected at runtime on a per-application (or per-core) basis. For this work we evaluate choosing among the 5 prefetching approaches listed in Table I.

Hardware Prefetching: For hardware prefetching we use the standard stream prefetchers available in our Intel SandyBridge processor. This processor supports four levels of prefetcher control: L2/LLC stream, L2/LLC adjacent cache line, L1 adjacent line, and L1 stream.

Software Prefetching: For software prefetching we use the technique described in [9], which first models an application's cache behavior to identify delinquent loads, and then inserts accurate software prefetches (including non-temporal prefetches) for such loads while avoiding cache pollution.

To enable and disable hardware prefetching we use each core's model specific registers. For controlling software prefetching we use the *Protean code* framework [12] . With Protean code we can create multiple versions of key loops in the code with and without software prefetching. Protean code then allows us to easily switch which version we execute at runtime, effectively turning software prefetching on and off with a very low overhead.

III. MODELING PREFETCHING IN APPLICATION MIXES

Our goal is to improve multicore performance by choosing the best individual prefetch setting for each application in a mix. For a mix of four applications with 5 prefetch options (Table I) this means identifying the best combination of prefetch settings from the 625 possible combinations. For larger core counts this number becomes even more intractable.

To address this, we first investigate how prefetching choices affect an application's off-chip bandwidth consumption and performance within a mix (Section III-A). From this investigation we discover that we can efficiently build a linear model for each application based only on the best and worst bandwidth scenarios for each prefetch setting (Section III-B). This reduces the number of combinations we need to explore from 625 (exponential in the number of applications) to 34 (linear in the number of applications). Using our new ability to model each application's behavior, we then develop an iterative solver that allows us to predict each application's performance and bandwidth in a mix for arbitrary prefetch settings, and evaluate the accuracy of our prediction against the baseline of running all 625 combinations (Section III-C) .

A. Impact of Prefetching on Applications in Mixes

To understand how prefetching choices affect an application's memory bandwidth consumption and performance we need to look into each application's sensitivity to prefetching (how much its performance increases as data is prefetched faster) and how effective each prefetching approach is for that application. These sensitivities are particularly important in application mixes because off-chip bandwidth and LLC are shared, and often scarce, resources [8, 9, 11]. To do so, we first investigate the relationship of individual applications' off-chip bandwidth consumption to that of the other applications executing in the mix, for each prefetch setting. This gives us insight into how off-chip bandwidth is shared within the mix, taking into account the aggressiveness of each application. To connect this to performance, we then investigate how the individual applications' performance changes as a function of their own off-chip bandwidth consumption, again, as for each prefetch setting. By looking at this data for all 625 possible combinations across 14 mixes we see that we can develop an efficient model for application-prefetch sensitivity (Section III-B).

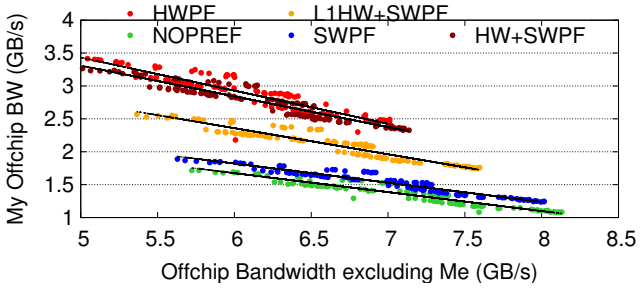


Figure 3. Off-chip bandwidth for *GemsFDTD* for the five different prefetch options as a function of the bandwidth of the other applications in the mix, across all 625 possible combinations in *mix-1*. For each prefetch option there is a clear linear relationship between *GemsFDTD's* bandwidth and the bandwidth demand (pressure) of the other applications in the mix (excluding GemsFDTD). The black lines show the linear BW-BW Model based on the reference data from all 625 possible prefetch combinations for this mix.

Bandwidth: To understand how applications share off-chip bandwidth in a mix as a function of prefetch settings, we look at the individual application off-chip bandwidth for

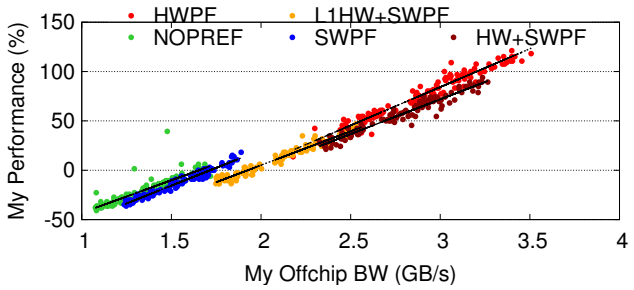


Figure 4. Performance (speedup over No-Prefetching only) of *GemsFDTD* for the five different prefetch options as a function its bandwidth, across all 625 possible combinations in *mix-1*. For each prefetch option there is a clear linear relationship between *GemsFDTD's* performance and its bandwidth. The black lines show the linear BW-Perf Model, based on the reference data from all 625 possible prefetch combinations for this mix.

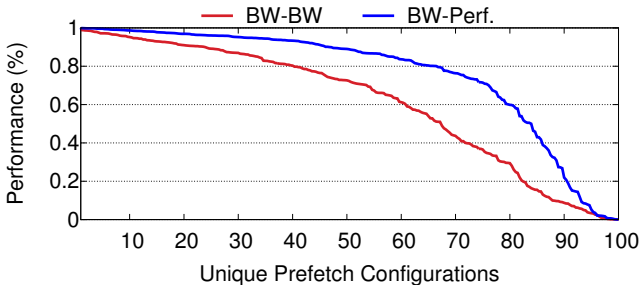


Figure 5. Distribution of the R^2 values of the linear model fit for all prefetch options across all cores in 14 mixes (shown in Table II). Higher score (closer to 1) shows a better fit of the model.

one application in the mix as a function of the bandwidth of the remaining applications in the mix. That is, how the share of off-chip bandwidth each individual receives changes as the other applications consume more bandwidth. Figure 3 shows the bandwidth for the individual application *GemsFDTD* (y-axis) for all 625 possible prefetch combinations in *mix-1* (Table II) as a function of the off-chip bandwidth consumed by the three other applications (x-axis), grouped by the prefetch choice for *GemsFDTD*.

From Figure 3 it is clear that there is a nearly linear relationship between how much bandwidth the individual application can consume relative to the remaining applications in the mix. The choice of prefetching method affects the absolute amount of bandwidth the application can get, with no prefetching (NOPREF) delivering the least and hardware prefetching (HWPREF) the most. We can use this data to develop linear models for how an individual application's bandwidth is affected by the bandwidth pressure it sees from the remainder of the mix. These *BW-BW Models* are plotted in black in Figure 3 for each prefetch setting.

Performance: In addition to understanding how applications share off-chip bandwidth as a function of prefetch setting, we need to understand the relationship between off-chip bandwidth and performance. Figure 4 shows the performance (speedup over the baseline of No-Prefetching across all cores) of *GemsFDTD* across all 625 combinations in *mix-1* as a function of its bandwidth and the prefetch setting. Here again we see a nearly linear relationship between off-chip bandwidth and performance, but the slope and offset varies depending on the prefetching choice. We model this effect using a linear regression (black lines in Figure 4) as the *BW-Perf Model*.

The accuracy of the BW-BW Model and BW-Perf Model are evaluated in Figure 5, which shows the R^2 fit for all individual applications across all prefetch choices in the 14 mixes (Table II). For reference, the R^2 fits for the models shown in Figures 3 and 4 are above 0.8.

With these models we now have the tools we need to predict performance for a mix of applications. To understand performance for an arbitrary combination of prefetch choices we can iteratively model an individual application's speedup and off-chip bandwidth, and then integrate that with the off-chip bandwidth and performance of the other applications in the mix (Section III-C). However, to make this approach practical, we need to develop a way to find the BW-BW Model and BW-Perf Model without requiring data from all 625 combinations.

B. Efficiently Modeling Prefetch Behavior

The data shown in Figures 3 and 4 indicate that we can model the application's off-chip bandwidth and performance behavior in a mix using a simple linear model for each prefetching choice. Further, with such models we can iteratively solve for how any combination of prefetching choices will perform in a mix. However, the data presented so far is based on exhaustive evaluations of all 625 possible combinations, and is neither practical to gather in itself (taking nearly 21 hours with our 2 minute-per-mix experiments) nor remotely feasible for larger core counts. To address this problem, we note that we can approximate the linear BW-BW Models and BW-Perf Models by points at either extreme: the best (most) off-chip bandwidth achieved for each prefetch setting and the worst (least). If we can efficiently find these points (or close approximations to them) then we can reduce the required data to two points per application per prefetch setting.

To identify the best and worst bandwidth points for each prefetch setting we make the following assumptions: The best conditions (most off-chip bandwidth) will occur when the other applications in the mix are using as little off-chip bandwidth as possible, which means setting them to *No Prefetching*, and the worst (least off-chip bandwidth) will occur when the other applications in the mix use as much off-chip bandwidth as possible, which means setting them to *Hardware Prefetching*. We abbreviate these settings as: *x-N-N-N*, with *No Prefetching* for the other applications in the mix, and *x-H-H-H* for *Hardware Prefetching* for the other applications in the mix.

The accuracy of this best case/worst case approximation is shown for each application in *mix-1* in Figure 6. The first column shows the BW-BW Model and the second column the BW-Perf Model. The results from the best case/worst case points are shown for each prefetch setting with colored dots and the linear fit with colored lines. The linear fit to all 625 combinations is shown with black lines for reference. Figure 6 demonstrates that the fit to the best case/worst case points is very close to the fit using all combinations, while only requiring gathering data for 34 combinations.

C. Multicore Performance Prediction

The BW-BW Model and BW-Perf Model allow us to understand the interaction between the off-chip bandwidth demands and allocations in a mix and the resulting performance for each application, as a function of the chosen prefetch settings. To determine the performance for an arbitrary selection of prefetching choices in a mix, we run an iterative solver that uses the BW-BW Model to find a steady-state solution to how much bandwidth each application receives, which then allows us to estimate performance from the BW-Perf Model.

For example, if we want to predict behavior for a mix of 4 applications with prefetch settings core 0: SWPF, core 1: L1HW+SW, core 2: HW+SWPF, and core 3: HWPF, we would collect data for the 4 best case settings (most bandwidth for the individual application) of *SWPF-N-N-N*, *N-L1HW+SW-N-N*, *N-N-HW+SWPF-N*, *N-N-N-HWPF* and the 4 worst case settings (least bandwidth for the individual application) of *SWPF-H-H-H*, *H-L1HW+SW-H-H*, *H-H-HW+SWPF-H*, *H-H-H-HWPF*. From this data we can create the four required BW-BW Models and BW-Perf Model models.

We then begin by using the four best case settings as the starting point for our iterations. We take the sum of the required individual off-chip bandwidth for each application and use the BW-BW Model to compute new off-chip bandwidths for each individual application based on the mix off-chip bandwidth. (e.g., How much bandwidth each individual application will receive given its prefetch settings and the other applications' bandwidth consumption.) We iterate until the total off-chip bandwidth converges to within 5% of the previous iteration. At this point we estimate the individual performance of each application from its estimated individual off-chip bandwidth using the BW-Perf Model. If we want to find the best combination of prefetch settings, we repeat the above for each setting for each application (625 in total), which requires 34 best-case/worst-case data points.

To evaluate the accuracy of this approach, we predicted the performance (weighted speedup) and off-chip bandwidth for all 625 combinations of prefetch settings for *mix-1* and compared them to the actual measured values in Figure 7. Prefetching combinations are sorted in decreasing order of

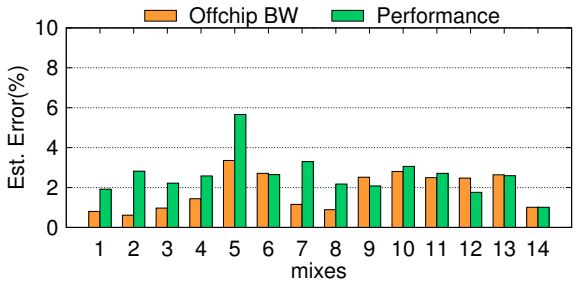


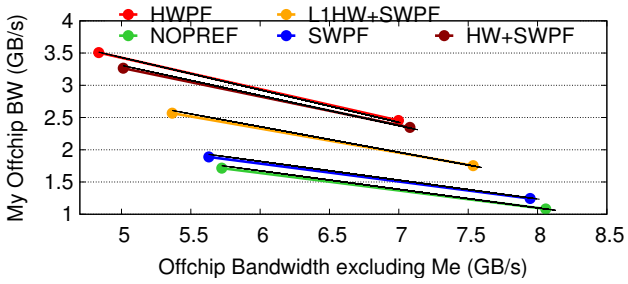
Figure 9. Average relative error in estimating Off-chip Bandwidth and Performance across all (625) prefetch combinations for 14 mixes. Our average prediction error for off-chip bandwidth is below 4% for all mixes, while performance is below 4% for all but one.

real performance, and a magnified view of part of the off-chip bandwidth data is shown on the side (Figure 7b). Similar data is shown for mix-14 in Figure 8b. Here the clear shift in performance after the best 380 configurations is correctly captured by our model. The average errors for predicted performance and total off-chip bandwidth are 2% and less than 1%, respectively. Accuracy numbers are shown for all mixes in Figure 9, and estimation errors are below 3.5% for all mixes except mix-5, which is 6%. This data shows that using best case/worst case input data and our iterative solver can predict performance and bandwidth for arbitrary combinations of prefetch settings.

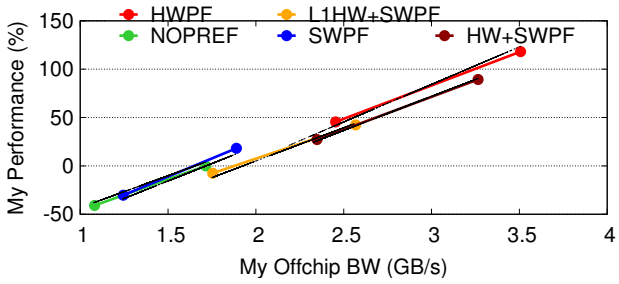
However, high average accuracy may not be sufficient to make good choices. As can be seen in Figure 7, our performance prediction for the best combination (leftmost point) is actually lower than for a slightly lower performance point directly to its right. An algorithm or runtime that used these models to choose prefetcher settings might choose the wrong combination in this case. To evaluate this, we look at the difference in performance from choosing the best option among the top 5 of the 625 possible combinations Section IV and show that we can make robust decisions even in the presence of this uncertainty.

IV. Evaluation

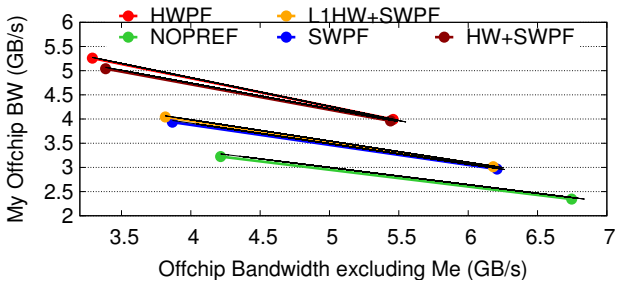
In Section III we developed a simple, yet efficient and accurate model for predicting performance and bandwidth for mixes of applications with arbitrary per-application prefetch settings. To demonstrate the value of this capability, we use it to optimize the performance of mixed workloads on a commodity 4-core processor (Intel Sandybridge, i7-2600K, all cores at 3.4GHz, 32kB/256kB private caches, 8MB shared cache, DDR3-1333, stream peak bandwidth measured at 13 GB/s). We limit ourselves to evaluating workloads on 4 cores as the overhead of gathering reference performance results for more cores grows prohibitively. (Our reference data for a single mix was collected running each setting for two minutes and took 21 hours for 4 cores. Collecting reference data for 8 cores would take over 18



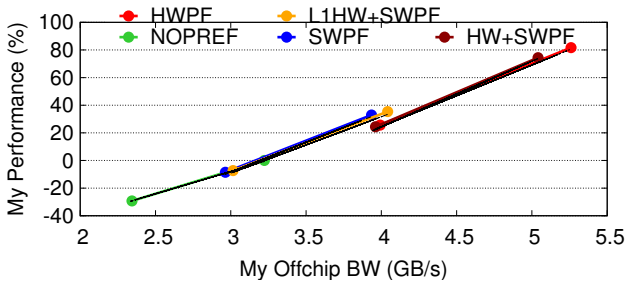
(a) Application Off-chip Bandwidth Model - *GemsFDTD*



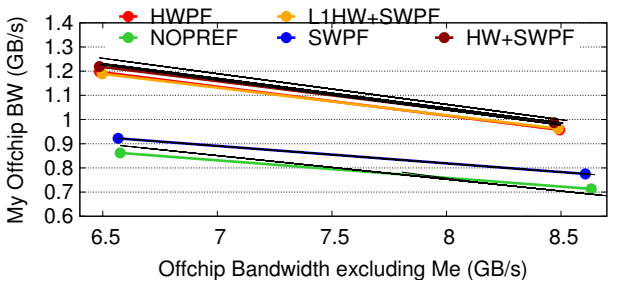
(b) Application Performance Model - *GemsFDTD*



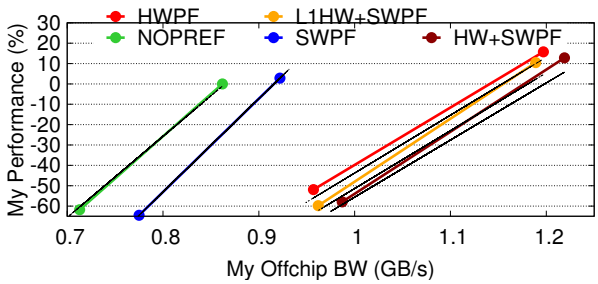
(c) Application Bandwidth Model - *lbm*



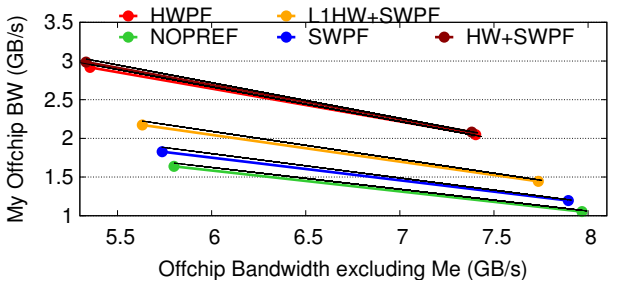
(d) Application Performance Model - *lbm*



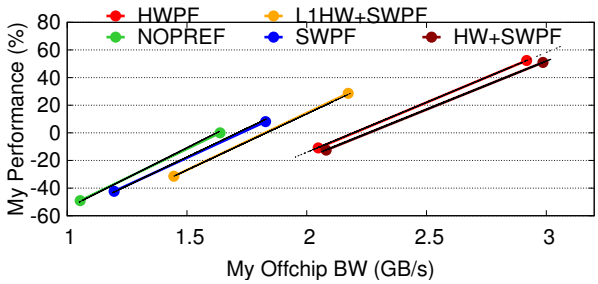
(e) Application Bandwidth Model - *astar*



(f) Application Performance Model - *astar*

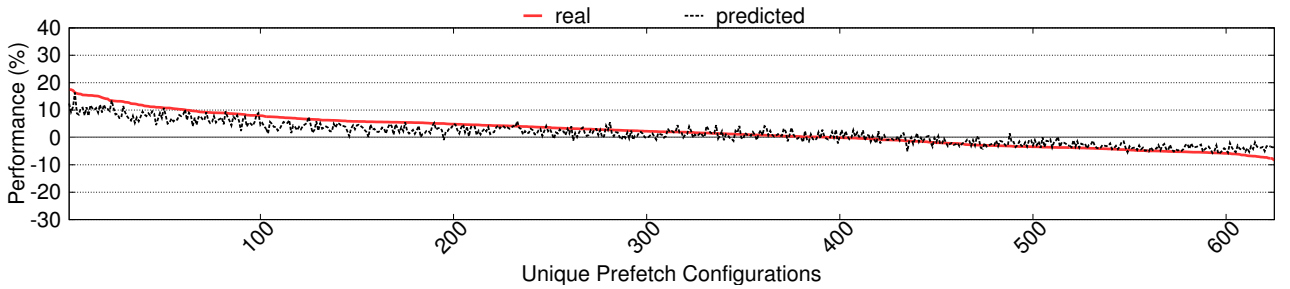


(g) Application Bandwidth Model - *soplex*

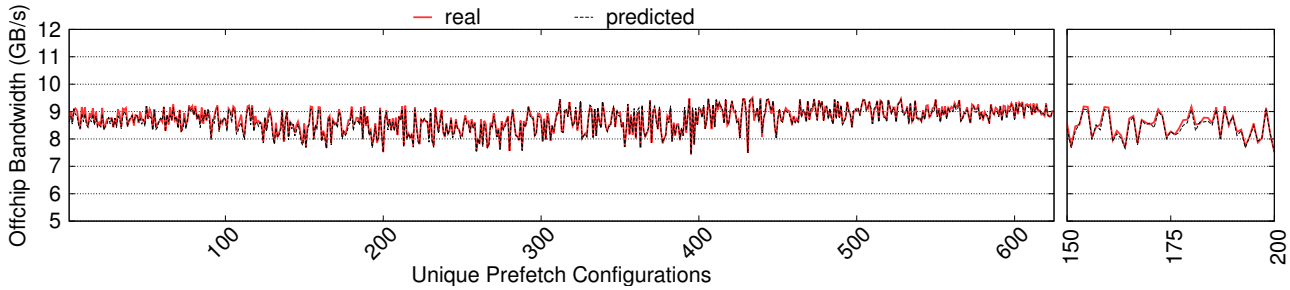


(h) Application Performance Model - *soplex*

Figure 6. Comparison of our Off-chip Bandwidth Model (left column, individual application bandwidth vs. system bandwidth excluding the individual application) and Performance Model (right column, individual speedup vs. individual off-chip bandwidth) for the 4 applications in *mix-1*. Colored points and lines show the result of modeling behavior using only the best case (*x-N-N-N*) and worst (*x-H-H-H*) points discussed in Section III-B, while black lines show the results of using the full 625 combinations. The model created using the extreme points accurately depicts the overall off-chip bandwidth and performance trends for the applications for each prefetch choice, as can be seen by the very good match between the black and colored lines for most applications.



(a) Performance Prediction



(b) Off-chip Bandwidth Prediction

Figure 7. Comparison of Performance prediction (top) and Off-chip Bandwidth prediction (bottom) across the all combinations of prefetch settings for mix-1. Off-chip Bandwidth prediction error is less than 1% on average for this mix. An enlargement of several of the mixes is shown to demonstrate how accurate the bandwidth prediction is.

months on an 8-core machine, or 625 8-core machines in parallel for 21 hours.) For our experiments, we generated 14 random mixes of applications from SPEC2006 [6] whose datasets do not fit in the LLC, as listed in Table II. All benchmarks were compiled using LLVM (version 3.3) with -O3 optimization.

Our approach is to use the method described in Section III-C to predict performance for all prefetching combinations based on the BW-BW Models and BW-Perf Models, and then choose the most promising combination. To evaluate these results, we ran each mix for 2 minutes and compare weighted speedup for a *baseline with no prefetching* to the following options:

- **Hardware Prefetching:** Standard hardware prefetching enabled across all cores.
- **Static Uniform Best (SUB):** The best performance of the 5 prefetch combinations where a single prefetch option is applied uniformly across all applications in the mix (Similar to AREP [11]).
- **Oracle:** The best performing prefetch combination from the 625 reference runs for each mix.
- **Perf-Insight:** The performance of prefetch combination that was *predicted* to have the best performance based on our models, trained using the 34 best case/worst case data points.
- **Perf-Insight top 5:** The best performing combination based on running the top 5 prefetch combinations predicted by our models, trained using the 34 best case/worst case data points. This evaluates the impact of the noise in selection of the best choice.

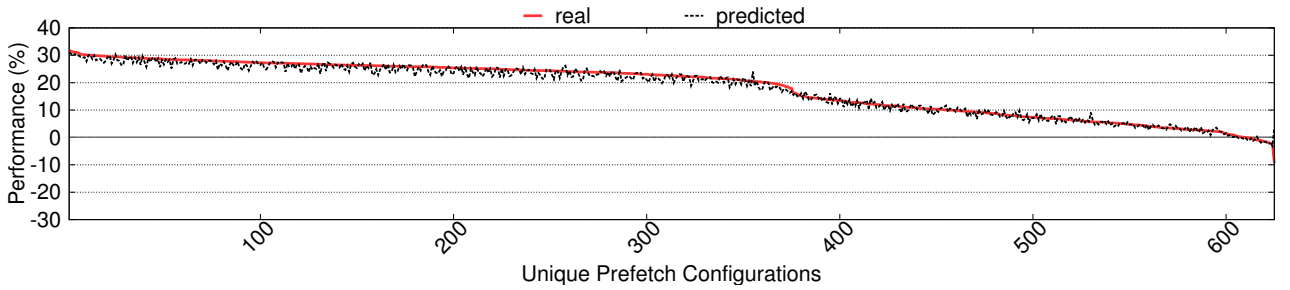
Mix-1	GemsFDTD, lbm, astar, soplex
Mix-2	GemsFDTD, lbm, libquantum, astar
Mix-3	GemsFDTD, lbm, soplex, libquantum
Mix-4	GemsFDTD, mcf, astar, soplex
Mix-5	astar, lbm, leslie3d, libquantum
Mix-6	astar, lbm, omnetpp, zeusmp
Mix-7	astar, libquantum, omnetpp, soplex
Mix-8	bwaves, bzip2, lbm, omnetpp
Mix-9	bwaves, leslie3d, mcf, zeusmp
Mix-10	bwaves, leslie3d, omnetpp, zeusmp
Mix-11	bwaves, milc, omnetpp, zeusmp
Mix-12	bwaves, omnetpp, xalan, zeusmp
Mix-13	bzip2, leslie3d, xalan, zeusmp
Mix-14	mcf, omnetpp, soplex, sphinx

Table II
THE 14 MIXES OF MEMORY-INTENSIVE SPEC2006 APPLICATIONS.

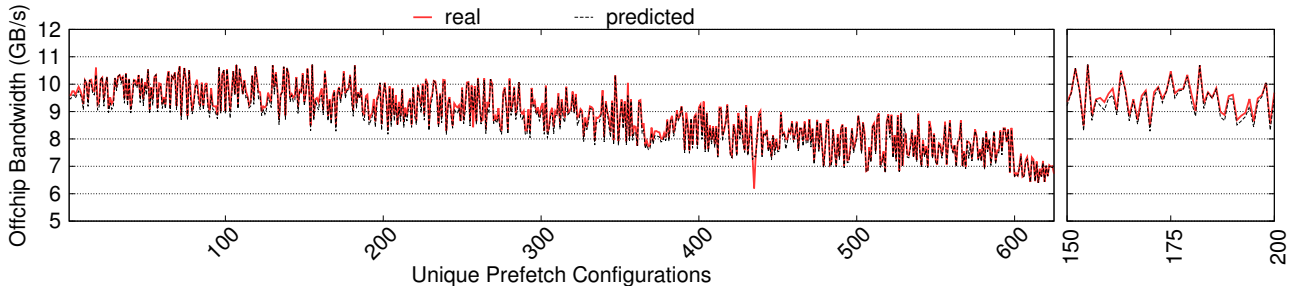
It is worth noting that randomly sampling the configuration space would be unlikely to deliver good results. For the 4-core case with 625 combinations, sampling 34 random configurations would be expected to yield one configuration in the top 20 entries on average, falling to 1 in the top 92 for 5 cores.

A. Performance and Off-chip Bandwidth

Figure 10 compares the weighted speedup we can achieve by changing prefetch settings across our 14 mixes. It is interesting to note that across the mixes hardware prefetching causes a slowdown in 3 of the mixes, and negligible



(a) Performance Prediction



(b) Off-chip Bandwidth Prediction

Figure 8. Comparison of Performance prediction (top) and Off-chip Bandwidth prediction (bottom) across the all combinations of prefetch settings for mix-14. Off-chip Bandwidth prediction error is less than 1% on average for this mix. An enlargement of several of the mixes is shown to demonstrate how accurate the bandwidth prediction is.

speedup in 3 more. The reason for hardware prefetching's poor performance can be seen in Figure 11, where the off-chip bandwidth for hardware prefetching is nearly always higher than the other choices. This observation is in agreement with Khan, Laurenzano, Mars, Hagersten, and Black-Schaffer [11], who demonstrated that when off-chip bandwidth demand with hardware prefetching exceeds approximately 70% of the system bandwidth it is beneficial to regulate prefetching to improve performance.

Compared to hardware prefetching's average 9% weighted speedup, the best we could achieve with per-application prefetch settings (Oracle) is a 22% speedup on average. It is interesting to note that this speedup actually comes with an overall reduction in bandwidth (Figure 11), as choosing per-application prefetch settings allows us to be more intelligent in how we tradeoff bandwidth and performance. Perf-Insight approach comes very close to the optimal results in nearly all mixes, with an average weighted speedup of 19% or 21% for the best of the top-5. The difference between Perf-Insight and Perf-Insight top-5 is small, with no mix seeing more than a 5% impact due to noise in our predictions. These results demonstrate that our model is both accurate and robust in identifying top-performing combinations. With Perf-Insight we are able to efficiently make prefetching choices that significantly speedup a wide range of mixes on commodity hardware.

Figure 12 shows how close each of the prefetching choices comes to choosing the fastest option (100th percentile, far left point in Figures 7a and 8a) for each mix. In 13 of the 14 mixes Perf-Insight is within 2% of the best performing option, while both AREP (SUB) and hardware prefetching are noticeably further behind. In particular, across 5 mixes (2, 5, 6, 8, and 11) more than 60% of the 625 prefetching combinations outperform hardware prefetching alone, further emphasizing the need to intelligently choose prefetching strategies. From this data we can see that Perf-Insight can robustly identify very competitive prefetching configurations.

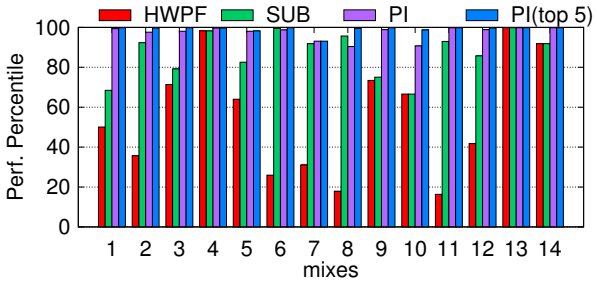


Figure 12. Performance percentile of the best predictions from Perf-Insight compared to best AREP (SUB) and hardware prefetching choices across 14 different mixes. Perf-Insight is generally better than 98th percentile, while hardware prefetching is significantly worse for many of the mixes compared to the optimal (100th percentile) configuration.

B. *Off-chip Data Volume*

Performance in a multicore is also affected by usage of the shared last-level cache. This can be a particular problem

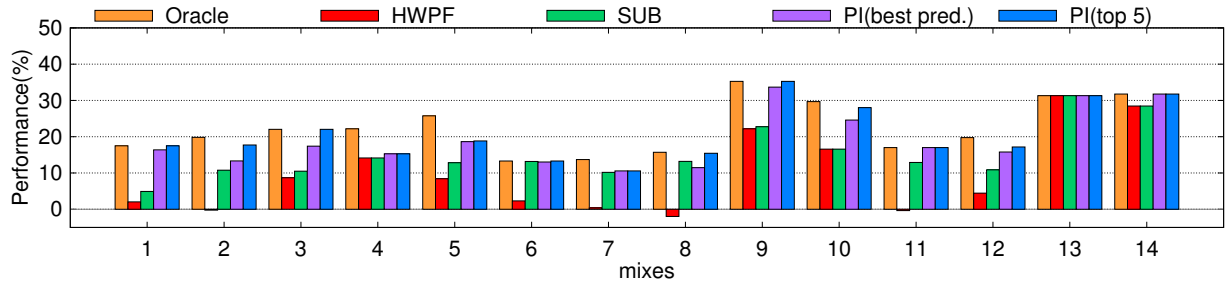


Figure 10. Weighted speedup of the different prefetching strategies for all 14 mixes relative to no prefetching. Hardware prefetching slows down 3 of the mixes, while Perf-Insight is very close to the per-application oracle. The robustness of the Perf-Insight predictions can be seen in the small deviation across the top 5 recommendations. The geometric mean speedups are: 9% for hardware prefetching, 22% for the oracle, 15% for SUB, 19% for Perf-Insight, and 21% for Perf-Insight-top-5.

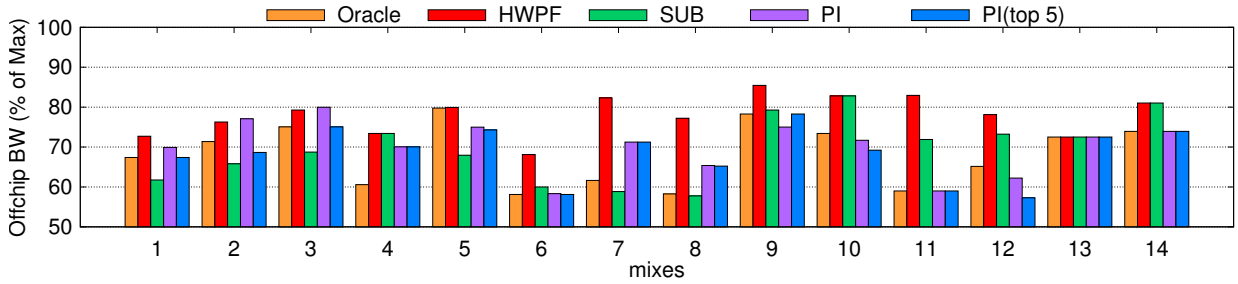


Figure 11. Off-chip bandwidth of the different prefetching strategies for all 14 mixes relative to no prefetching. The large increase in bandwidth due to hardware prefetching can be seen here (e.g., mix-1), while its corresponding lack of performance improvement is visible in Figure 10 (again, mix-1). The other prefetching options are much more effective at translating increased off-chip bandwidth into performance.

with aggressive prefetching if it brings in data that is not beneficial (either inaccurate or untimely), as it will evict other data from the LLC, which can hurt the performance of co-running applications. While it is difficult to isolate these effects in commodity systems, we can examine the increase in off-chip data *volume* to compare the efficiency of the different prefetch strategies. Increases in off-chip data volume can occur not just due to inaccurate prefetching strategies, but also as a side effect of application(s) running faster and evicting temporally useful data of other applications from the shared cache.

Off-chip data volume for for all 56 applications in our 14 mixes is shown in Figure 13, sorted in increasing order for each strategy. From this data we can see that hardware prefetching results in a far greater overall increase in off-chip data volume than the other strategies, which will directly hurt performance through shared cache resources.

C. Runtime Optimizations

The technique described in this work is fully amenable to runtime optimization of workloads. Switching hardware prefetching strategies at runtime is a simple matter of setting the model specific registers, with modern processors supporting reasonably fine-grain control over per-core prefetching, as we have demonstrated in this work. To enable and disable software prefetching efficiently, we can use

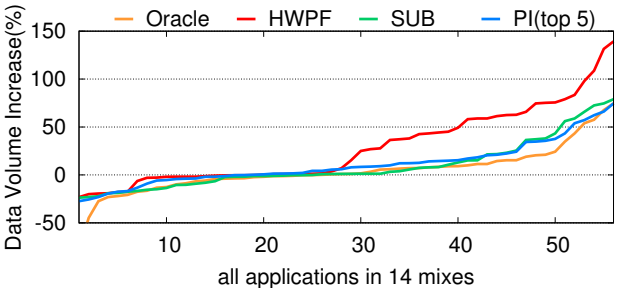


Figure 13. Off-chip data volume increase relative to to no prefetching for the final prefetching choices for each strategy across all 56 applications in the 14 mixes.

binary multi-versioning [12], which allows us to generate and then select different version of the binary at runtime. Khan, Laurenzano, Mars, Hagersten, and Black-Schaffer [11] implemented such a dynamic runtime for AREP and demonstrated that sampling prefetching combinations for 125ms was sufficient to judge performance.

With a 125ms per configuration overhead, it would take 4 seconds to gather the information needed to calibrate the Perf-Insight model for a 4-application mix. During the calibration phase, 3 of the 4 applications would be running with full hardware prefetching for half of the time (for the worst-case data collection) and for the other half 1

of 4 would be running with full hardware prefetching (for the best-case data collection). As a result, the performance loss for these 4 seconds compared to standard hardware prefetching would be minimal. Given the low overhead of such calibration, it would be very easy to periodically re-calibrate to adapt to program phase changes.

V. Related Work

Several works in recent years have proposed novel hardware prefetching schemes for improving the utilization of shared resources (off-chip bandwidth and shared cache capacity) in multicores [1, 2, 3, 7, 19, 23]. Liu and Solihin [14] have proposed analytical models for bandwidth partitioning to identify when prefetching can help improve system performance. However, their method is only useful in the context of the shared cache prefetchers in multicores and does not take into account multi-level prefetchers or software prefetching. Several other works have used software prefetching effectively to improve single-thread performance [15, 16, 17, 20, 24, 25, 28]. The list of general prefetching work is too extensive to cover in detail, so we discuss only the most relevant prior work.

The most closely related work is the Adaptive Resource Efficient Prefetching (AREP) [11] discussed earlier. AREP uses a brute-force exploration to choose one prefetch strategy for all applications, and could get away with it because the number of combinations was so small. Perf-Insight goes beyond this approach by developing models and an interactive solver to predict the performance of all possible prefetching combinations from only a few calibration points. With this model-based approach, Perf-Insight is able to achieve better performance on 4-core mixes and can scale up to larger systems.

Khan, Sandberg, and Hagersten [9] developed a resource-efficient software prefetching method to scale performance in multicores when shared resources are constrained. They showed that by using software prefetching (with cache-bypass hints) instead of hardware prefetching, performance scales better on fully-loaded systems. However, their method relies entirely on software prefetching and does not make any use of hardware prefetchers. Software-only prefetching is not optimal in all cases and can be improved significantly by judiciously applying hardware prefetching to the right applications in a mix.

Lee, Kim, and Vuduc [13] investigated combining hardware prefetching and software prefetching for single-threaded applications, concluding that caution should be exercised when mixing the two. In contrast to their work we use combination of hardware and software prefetching and apply this option to only individual applications in a mix that benefit from it.

Jiménez, Gioiosa, Cazorla, Buyuktosunoglu, Bose, and O'Connell [7] implemented a runtime mechanism for exploring different hardware prefetcher configurations on a POWER7 processor to maximize performance. The POWER7 processor allows the prefetcher aggressiveness to be adjusted between 7 different levels. Their runtime approach explores the best hardware prefetcher option on per-core basis (for two cores only), adopting the one that performs best. Unlike our work, they do not model and predict performance for the individual applications and they explicitly disable software prefetching.

Zhang, Laurenzano, Mars, and Tang [27] in their work SMiTe [27] describe a method to identify how applications co-running on the same core via SMT conflict with each other for the core's internal shared resources (such as the execution units). Based on application behavior profiles, their model can predict how any arbitrary combination of different applications perform if run together on the same core via SMT.

Sandberg, Black-Schaffer, and Hagersten [21] predict the impact of cache sharing on the performance of individual applications in mixes. Their methods profile applications' shared cache usage and performance under varying levels of shared cache pressure. Based on the individual profiles of each application, the method can predict performance for any arbitrary mix of applications, but only with regards to the shared cache. Sandberg, Sembrant, Hagersten, and Black-Schaffer [22] further investigated the impact of varying phase behavior.

VI. Conclusions

This paper introduces Perf-Insight, which delivers significant performance improvements of 21% (vs. 9% for the default hardware prefetching) for multicore workloads on commodity hardware by adjusting prefetcher settings on a per-application basis.

To achieve these improvements, we introduced a new, simple model for understanding the interaction between applications' individual performance and off-chip bandwidth, the off-chip bandwidth demand of the other applications in the mix, and the per-application prefetching choice. To make this approach practical, we developed a technique to efficiently calibrate the models using the best-case/worst-case settings, which allows us to both calibrate applications while running in the mix and also scale the calibration procedure linearly with the number of applications.

While we demonstrated Perf-Insight with a collection of 5 hardware and software prefetching techniques and chose the option with the best performance, the Perf-Insight approach works with any combination of prefetching choices that can be controlled at runtime. With Perf-Insight we are able to robustly choose very effective prefetch combinations for a range of workload mixes, and demonstrate significant real-world performance benefits across a wide variety of workload mixes on commodity hardware.

References

[1] J. D. Collins, H. Wang, D. M. Tullsen, C. Hughes, Y.-F. Lee, D. Lavery, and J. P. Shen. Speculative Precomputation: Long-range Prefetching of Delinquent Loads. In *ISCA*, 2001.

[2] E. Ebrahimi, O. Mutlu, and Y. Patt. Techniques for bandwidth-efficient prefetching of linked data structures in hybrid prefetching systems. In *HPCA*, 2009.

[3] E. Ebrahimi, C. J. Lee, O. Mutlu, and Y. N. Patt. Prefetch-aware shared resource management for multicore systems. In *ISCA*, 2011.

[4] D. Eklov, N. Nikoleris, D. Black-Schaffer, and E. Hagersten. Cache pirating: Measuring the curse of the shared cache. In *ICPP*, 2011.

[5] D. Eklov, N. Nikoleris, D. Black-Schaffer, and E. Hagersten. Bandwidth bandit: Quantitative characterization of memory contention. In *CGO*, 2013.

[6] J. L. Henning. SPEC CPU2006 benchmark descriptions. *SIGARCH Computer Architecture News*, 2006.

[7] V. Jiménez, R. Gioiosa, F. J. Cazorla, A. Buyuktosunoglu, P. Bose, and F. P. O'Connell. Making data prefetch smarter: Adaptive prefetching on power7. In *PACT*, 2012.

[8] M. Khan and E. Hagersten. Resource conscious prefetching for irregular applications in multicores. In *ICSAMOS*, 2014.

[9] M. Khan, A. Sandberg, and E. Hagersten. A case for resource efficient prefetching in multicores. In *ICPP*, 2014.

[10] M. Khan, A. Sandberg, and E. Hagersten. A case for resource efficient prefetching in multicores. In *ISPASS*, 2014.

[11] M. Khan, M. A. Laurenzano, J. Mars, E. Hagersten, and D. Black-Schaffer. Arep: Adaptive resource efficient prefetching for maximizing multicore performance. In *PACT*, 2015.

[12] M. Laurenzano, Y. Zhang, L. Tang, and J. Mars. Protean code: Achieving near-free online code transformations for warehouse scale computers. In *MICRO*, 2014.

[13] J. Lee, H. Kim, and R. Vuduc. When Prefetching Works, When It Doesn't, and Why. *ACM TACO*, 9 (1), Mar. 2012.

[14] F. Liu and Y. Solihin. Studying the impact of hardware prefetching and bandwidth partitioning in chip-multiprocessors. In *SIGMETRICS*, 2011.

[15] C.-K. Luk, R. Muth, H. Patil, R. Weiss, P. G. Lowney, and R. Cohn. Profile-Guided Post-Link Stride Prefetching. In *ICS*, 2002.

[16] C.-K. Luk, R. Muth, H. Patil, R. Cohn, and G. Lowney. Ispike: A Post-link Optimizer for the Intel Itanium Architecture. In *CGO*, 2004.

[17] J. Mars and R. Hundt. Scenario Based Optimization: A Framework for Statically Enabling Online Optimizations. In *CGO*, pages 169–179, 2009.

[18] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa. Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations. In *MICRO*, MICRO-44, 2011.

[19] S. Pugsley, Z. Chishti, C. Wilkerson, P. fei Chuang, R. Scott, A. Jaleel, S.-L. Lu, K. Chow, and R. Balasubramonian. Sandbox prefetching: Safe run-time evaluation of aggressive prefetchers. In *HPCA*, 2014.

[20] R. M. Rabbah, H. Sandanagobalane, M. Ekpanyapong, and W.-F. Wong. Compiler Orchestrated Prefetching via Speculation and Predication. In *ASPLOS*, 2004.

[21] A. Sandberg, D. Black-Schaffer, and E. Hagersten. Efficient techniques for predicting cache sharing and throughput. In *PACT*, 2012.

[22] A. Sandberg, A. Sembrant, E. Hagersten, and D. Black-Schaffer. Modeling performance variation due to cache sharing. In *HPCA*, 2013.

[23] C.-J. Wu, A. Jaleel, M. Martonosi, S. C. Steely, Jr., and J. Emer. Pacman: Prefetch-aware cache management for high performance caching. In *MICRO*, 2011.

[24] Y. Wu. Efficient Discovery of Regular Stride Patterns in Irregular Programs and Its Use in Compiler Prefetching. In *PLDI*, 2002.

[25] Y. Wu, M. J. Serrano, R. Krishnaiyer, W. Li, and J. Fang. Value-Profile Guided Stride Prefetching for Irregular Code. In *International Conference on Compiler Construction (CC)*, 2002.

[26] H. Yang, A. Breslow, J. Mars, and L. Tang. Bubble-flux: Precise online qos management for increased utilization in warehouse scale computers. In *ISCA*, 2013.

[27] Y. Zhang, M. Laurenzano, J. Mars, and L. Tang. Smite: Precise qos prediction on real-system smt processors to improve utilization in warehouse scale computers. In *MICRO*, 2014.

[28] Q. Zhao, R. Rabbah, S. Amarasinghe, L. Rudolph, and W.-F. Wong. Ubiquitous Memory Introspection. In *CGO*, 2007.