Boosting the Bouquet of Instruction Pointer Classifier-based Data Prefetching: What Works and What Doesn't?

Neelu Shivprakash Kalani * and Biswabandan Panda +

* Indian Institute of Technology, Kanpur
+ Indian Institute of Technology, Bombay

1 Introduction

State-of-the-art data prefetchers are competitive in providing performance improvement [1] [2] [3] [4], with Instruction Pointer Classifier-based Spatial Hardware Prefetching (IPCP) [1] providing the highest performance improvement. However, IPCP is not the best performing prefetcher in all scenarios, and still has unexplored potential for performance improvement. In this report, we discuss the following limitations of IPCP, along with ways to rectify these limitations: (i) late prefetching, (ii) cases where other state-of-the-art data prefetchers perform better than IPCP, (iii) low performance improvement in irregular benchmarks. For each section, we describe several experiments, and summarize with the results of the experiments, whether they succeed in eliminating the limitation or not, along with brief future direction in each domain. We use the ChampSim [5] simulator, enhanced with detailed virtual memory support for these experiments, and show the results for 50 million warmup instructions and 100 million simulation instructions, unless specified otherwise.

2 Lateness in IPCP

One of the limitations of IPCP is late prefetching. This means that, even though IPCP is able to learn the access pattern of the application, it is not able to issue timely prefetches corresponding to the processor's demand requests. This leaves unexploited potential for performance improvement that the IPCP data prefetcher can provide. Figure 1 shows the percentage of late prefetch requests that IPCP makes, specific to each class in IPCP. We see that the constant stride (CS) class of IPCP shows the highest amount of late prefetch requests among all classes. On average, 7.7% of the prefetches that IPCP makes are late. We observe the highest percentage of late requests for *mcf-782B* to be 26.23%. For computing late requests, we capture the number of demand requests that get hits in the miss status holding registers (MSHR) of the cache, due to in-flight prefetch requests. This indicates that the prefetcher predicts the demand requests correctly, but is unable to fetch the data on time for the demand request, and hence the demand request gets a hit in the MSHR. The prefetcher can convert these demand requests into cache hits if the prefetcher issues them earlier.

There are several approaches that can issue the prefetch requests on time and reduce the percentage of late prefetch requests. We now discuss and evaluate some of them.

Prefetch distance: A trivial attempt at making timely prefetch requests is to prefetch

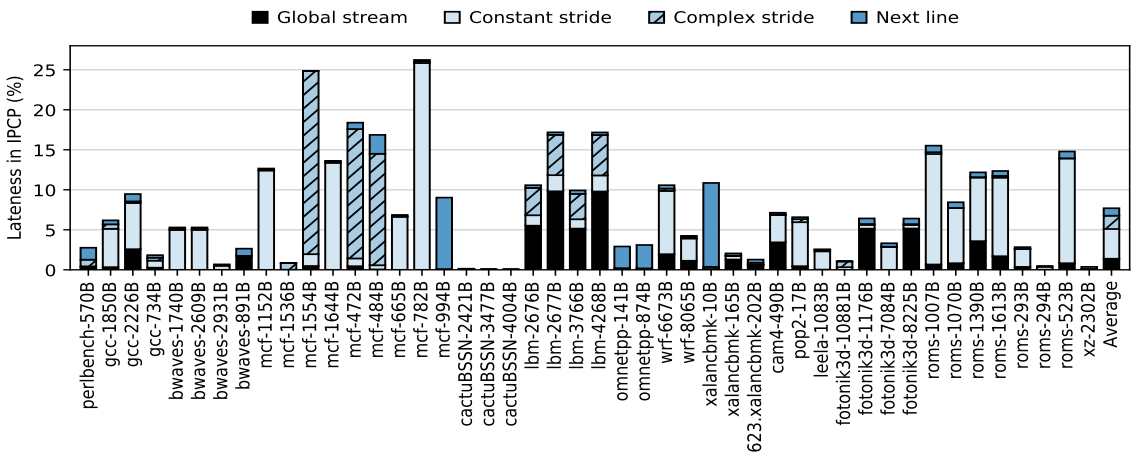

Figure 1: Percentage of late prefetch requests that IPCP makes (per class) for memory-intensive SPEC CPU 2017 benchmarks.

further ahead in the stream. Prefetchers target the demand requests likely to occur at the cache shortly i.e., before the outer cache level can process the prefetch request and send a response. However, if the prefetcher prefetches at a distance (demand requests further in the access stream), it can target demand requests that are likely to occur after a certain time, providing the memory hierarchy enough time to process and send a response. In the case of IPCP, we find that prefetching at a distance degrades performance on average, as Figure 2 shows, increasingly with increasing prefetch distance (going from 1 to 10). This is because the prefetch distance eliminates the prefetch requests that otherwise contribute to the application's overall performance improvement. For example, suppose there is an access to address A, and the prefetch stride is two. Without prefetch distance, the prefetch address would be A+2. But with a prefetch distance of one, the prefetch address becomes A+4. It is possible that the request for address A+4 is not useful to the processor or gets into the memory hierarchy before the demand request to address A+2, blocking some of the resources (eg. MSHR entries) for a request that corresponds to an older instruction in the processor pipeline. There are a few exceptions where prefetch distance improves performance (eg. bwaves-1740B, bwaves-2609B, mcf-1554B, and wrf-8065B). However, as it degrades performance in many other cases, the performance degrades overall.

Temporal correlation: Existing works [6] [7] [8] discuss that instruction sequences are highly likely to recur in the same order and leverage this observation to trigger timely prefetching. We explore such a correlation-based approach to improve timeliness in IPCP. Figure 3 shows the load IPs and corresponding load addresses in an instruction stream. Here, we see that IP B is the fourth load that follows IP A. Leveraging the fact that this sequence is likely to recur in the same manner, correlating these two IPs and triggering prefetching for IP B when IP A occurs can help reduce prefetch lateness. This makes IP A a trigger IP to trigger timely prefetching for target IP B. The data prefetcher also needs the load address corresponding to IP B to perform prefetching. When A occurs, we do not have this address. So, we store the last address that target IP B accesses and use it for prefetching. However, a load IP can access different addresses every time it recurs. Due to this, prefetching becomes

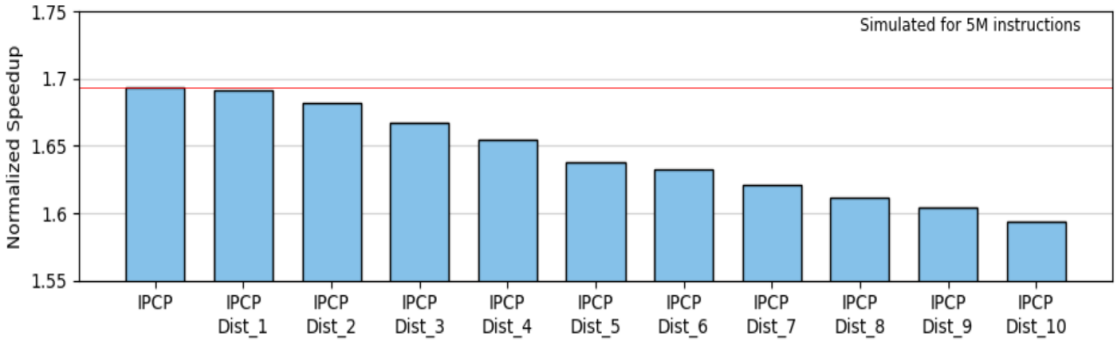

Figure 2: Performance improvement of IPCP prefetching at a prefetch distance varying from 1 to 10, across 46 memory-intensive SPEC CPU 2017 benchmarks.

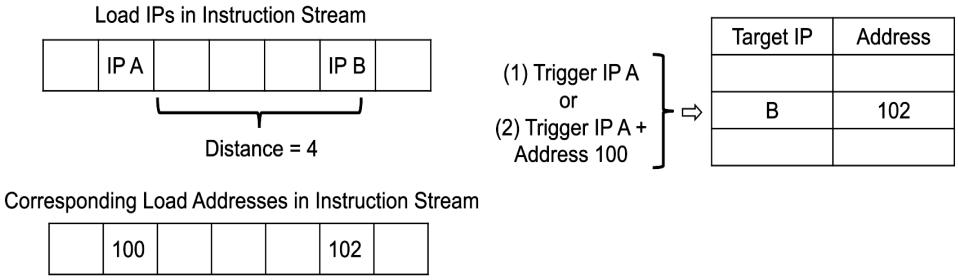

Figure 3: Correlating loads in temporal order for timeliness.

inaccurate and degrades performance with this scheme. Further, we explore the possibility that the load IP B accesses a certain address, 200, when the load IP A accesses the address 100. This means that, correlating an (IP+Address) pair as trigger to another (IP+Address) pair as target may be more accurate than correlating (IP) as trigger to an (IP+Address) pair as target. Even in this case, we see performance degradation in SPEC CPU 2017 benchmarks, as Figure 4 shows, while varying the distance from 2 to 10. CATCH [7] explores the concept of learning a delta-based correlation between a trigger IP and a target IP. However, this specific prefetcher component contributes to only 28% of the performance improvement that CATCH shows, and the rest comes from other members of the family of timely prefetchers that CATCH proposes. So, we do not include using this technique to improve timeliness in IPCP in the scope of this report.

TLB-directed prefetching for timeliness: For this study, we explore the address translation requests that miss in the first level data TLB (translation lookaside buffer) or DTLB, but hit in the second level TLB (STLB), to guide the data prefetcher for making timely prefetch requests. On STLB hits (87% STLB hit rate for SPEC CPU 2017 benchmarks), we use the state-of-the-art L2 data prefetcher i.e., SPP (Signature Path Prefetching) [3] [1] to perform timely prefetching into L1D in the presence of the state-of-the-art L1D prefetcher IPCP. This is done because of the close proximity of the STLB and the L2 on the

[1] We use an aggressive version of SPP as described in Section 3.

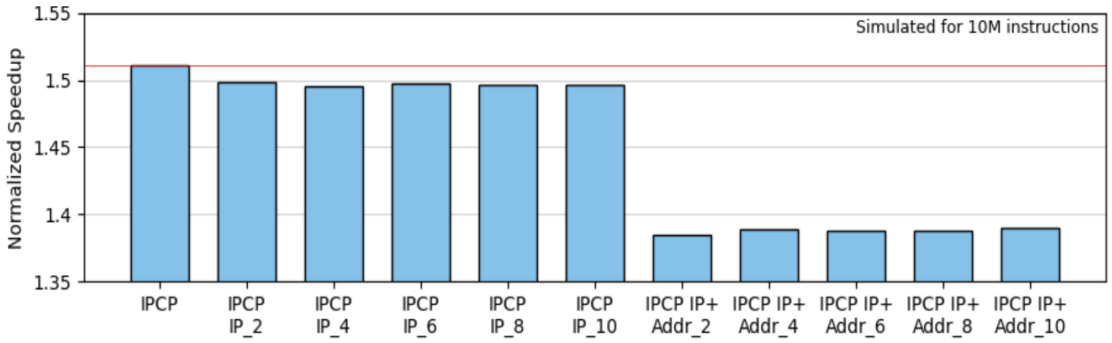

Figure 4: Performance improvement of correlation-based timely prefetching for IPCP, with trigger as IP or IP+Address, and correlation distance varying from 2 to 10, across 46 memory-intensive SPEC CPU 2017 benchmarks.

Table 1: TLB-directed prefetching compared with normal prefetching.

Metric	With TLB-directed prefetching
Performance	-11.01%
L1D prefetcher accuracy	-10.55%
L1D prefetches issued to next cache level	170%
L1D late prefetches	33.94%

processor chip. Thus, on every STLB hit, a hint is sent to the L2 prefetcher,incorporated as a single bit in the unused bits of the physical address that the L2 prefetcher receives. The L2 prefetcher generates prefetch requests corresponding to the demand request. When these prefetch requests fill into the cache, the L2 prefetcher uses the hint from STLB (carried forward into the prefetch address that the L2 prefetcher generates) to trigger prefetches from the L1D. We enable this by sending a packet from L2 to L1D that contains the prefetch address, so the L1D can issue the prefetch request and follow the usual route that L1D prefetches take. As Table 1 shows, with this scheme, we observe a performance loss of 11%. This is because, as L1D is very small, filling all the L2 prefetches in L1D leads to pollution. This is evident by L1D prefetcher accuracy reducing by 10.55% and prefetches issued to next cache level increasing by 170%. Instead of improving timeliness, this scheme ends up increasing L1D late prefetches by 33.94%, as the increase in prefetch requests also leads to contention at prefetch queue. So, instead of filling all the L2 prefetches into L1D, we try allowing this only when the L2 prefetcher's accuracy is higher than 90%. However, we still observe an average performance degradation of 0.2% across memory-intensive SPEC CPU 2017 benchmarks.

Prefetcher aggressiveness and cross-page prefetching: Another technique to improve the timeliness of prefetch requests is to increase the prefetcher's aggressiveness i.e., prefetch degree [9]. Consider the following example: the cache receives demand requests for address A, and then A+2 (access pattern with stride of two). On the demand request for address A, a prefetcher with prefetch degree of three prefetches the blocks for addresses

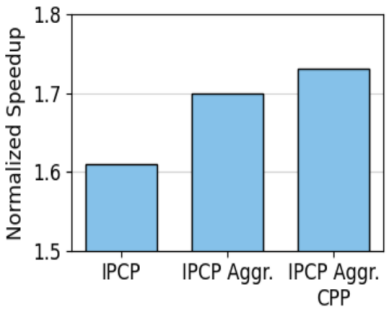

Figure 5: Performance improvement with IPCP, IPCP aggressive, and IPCP aggressive with cross-page prefetching (CPP) at L1D, across 46 memory-intensive SPEC CPU 2017 benchmarks.

A+2, A+4, A+6. On the demand request for address A+2, the prefetcher then prefetches for addresses A+4, A+6, A+8. Instead, if we increase the prefetch degree to five, the demand request to A can trigger the prefetches for addresses A+2, A+4, A+6, A+8, A+10. This can ultimately lead to timely prefetches and help in improving performance.

In the case of a physically-indexed physically-tagged (PIPT) cache, the prefetcher cannot issue prefetches beyond the page size, as the OS can map adjacent pages in the virtual memory of a process to non-adjacent pages in the memory. However, with a virtually-indexed physically-tagged cache, the prefetcher can issue prefetches beyond the page size using virtual addresses. We use a simulator with extensive virtual memory support, and direct the translation requests corresponding to L1D cache to the second-level translation look-aside buffer (TLB). This allows the prefetcher to prefetch across pages (cross-page prefetching) and helps in making the prefetcher timely.

Figure 5 shows that increasing the aggressiveness of IPCP improves performance by 8.9%. The degree of global-stream class is increased from 6 to 16, degree of all other classes are doubled. With cross-page prefetching, the performance improves further by 3.1% on top of making the prefetcher aggressive.

Though, we would like to note that the simulator we use [5] does not simulate the effects of the processor exploring the wrong execution path in case of mispredicted branches, leaving less room for pollution in the caches. An aggressive prefetcher may also cause pollution in the cache by prefetching for the wrong execution path. However, IPCP has a degree throttling mechanism which reduces the prefetch degree when the prefetcher accuracy is lower than a certain threshold. Thus, the deep speculation that cross-page prefetching allows may still be a source of performance improvement.

Piggyback prefetching: Making the prefetcher aggressive can increase the prefetcher's timeliness and hence, lead to performance improvement. However, since the L1D is small, it is not feasible to increase the prefetcher's aggressiveness beyond an extent without the risk of polluting the small L1D. However, the L2 is about 10 times larger than the L1D. So, the prefetches predicted to be accessed further in the access stream can be prefetched till L2, instead of L1D. Thus, with prefetch requests from IPCP at L1D, we send one bit to denote whether to piggyback, 2 bits to denote the class of IPCP, and 4 bits to denote the prefetch degree, in the form of metadata, similar as IPCP already does for prefetching at L2.

We perform piggyback prefetching for both CS and GS classes of IPCP on top of aggressive cross-page prefetching that we describe earlier. Piggyback prefetching at L2 prefetches for the same prefetch degree that L1D uses for prefetching. With this, we observe a performance improvement of 1.61% on average, across 46 memory-intensive SPEC CPU 2017 benchmarks.

What works and what doesn't? The following summarizes the schemes that we experiment with, to improve IPCP's timeliness: (i) prefetching at a distance degrades performance on average, (ii) using temporal correlation between (IP, IP+Address) or (IP+Address1, IP+Address2) do not improve performance, rather a scheme to correlate (IP+Address1, IP+Address1+Delta) could be a promising direction as per [7], (iii) using TLB-directed hints to prefetch at L1D leads to pollution and degrades performance, and (iv) increasing the prefetcher's aggressiveness along with allowing prefetching across pages improves performance by 12% on average, (v) finally, piggyback prefetching the requests predicted to be further in the access stream, and filling them till L2 further improves performance by 1.61% on average, on top of aggressive cross-page prefetching.

Future direction: The following describes the future research direction in this domain: (i) A temporal correlation scheme that correlates two IPs with a delta-based correlation, as CATCH [7] uses in its TACT-cross prefetcher component, can be used to trigger timely prefetching, instead of using last address, (ii) Instead of using a fixed correlation distance, a dynamic correlation distance can be determined based on the time it takes to serve the request [6] [10].

3 Pushing the Performance of IPCP

State-of-the-art data prefetchers [3] [2] are competitive with IPCP in terms of performance improvement. However, these data prefetchers do not show the same performance behaviour per benchmark. This means that, for several benchmarks like *mcf*, *cactuBSSN*, *lbm*, *xalancbmk*, and *roms*, SPP (i.e., a region-based delta prefetcher) performs better than IPCP. Whereas, for certain benchmarks like *gcc*, *bwaves*, *mcf*, *omnetpp*, *wrf*, and *fotonik3d*, Bingo (i.e., a pattern history based prefetcher which triggers accurate prefetching on long events) performs better than IPCP. This shows a scope for enhancing IPCP to improve performance to match it with other state-of-the-art data prefetchers for all benchmarks. Keeping this in mind, we perform some experiments for improving performance of IPCP as described below.

Large region training: IPCP performs training for each class within 4 KB regions. Each 4 KB region contains 64 blocks of data, each of 64 bytes. This means that, if an application has a stride of greater than 64, IPCP is unable to learn it for stride-based classes. We find that, 93.2% of the time, performing 4 KB region learning is enough to capture strides. However, we experiment with IPCP training on large region sizes, varying from 4 KB to 1 MB, to find the performance scope. On average, we find that, no region size performs better than 4 KB region size. However, certain benchmarks benefit from training on region sizes larger than 4 KB, as Figure 6 shows. We also perform large region training per class in IPCP, i.e., large training for one of the classes while the other classes train on 4 KB region size. We find that CS class of IPCP benefits most from large region training, closely followed by CPLX class. GS class in IPCP does not benefit from large region training, as it takes longer to train for a global stream to identify large regions as densely accessed ($> 75\%$ of the region

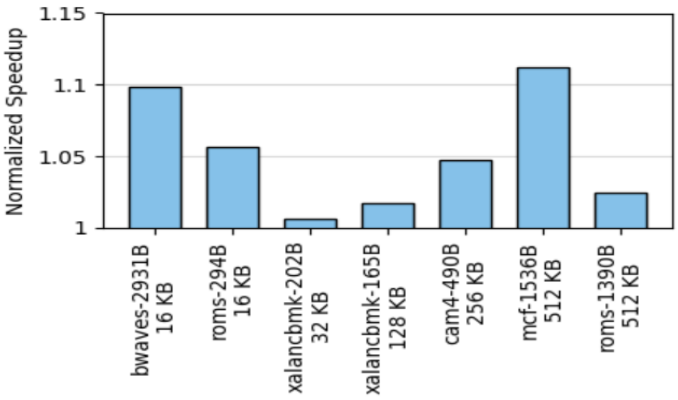

Figure 6: Performance improvement of large region training in IPCP over 4 KB region training (aggressive and cross-page prefetching enabled).

is accessed).

Region-based prefetching in IPCP: Since IPCP is an IP-based classifier, it doesn't have a region-based prefetching component. However, we find that state-of-the-art data prefetchers i.e., SPP (region-based) and Bingo (which uses both IP and data address as signature) perform better than IPCP for some benchmarks. So, we introduce a region-based prefetching component at L2 along with IPCP at L1D and SPP at L2. We use an aggressive version of SPP where we reduce the SPP fill threshold to 25 and prefetch threshold to 15 as it performs better than the default threshold values. In this scheme, we perform 2 KB region-based training by capturing the lines that the application requests for, within a 2 KB region, in the form of a bit-vector (32 bits), for 72 regions. We perform prefetching for all the set bits in the bit-vector whenever accesses to same region arrive again. This is done only for regions that get accesses to at least 1/4th of the region. This improves performance by 1.3% on average across 46 memory-intensive SPEC CPU 2017 benchmarks. Further, instead of prefetching only the set bits, we prefetch the whole sub-region between the first and the last line accessed before as per the bit-vector. This improves performance by 1.8% on average over IPCP at L1D combined with SPP (tuned) at L2.

(IP+Offset)CP: Instead of using shorter events like the recurrence of an IP, Bingo [2] uses longer events like recurrence of IP+Address or IP+Offset to trigger accurate prefetching. However, IPCP trains and triggers prefetching only for the shorter event i.e., recurrence of an IP. To find the utility of each of the events, IP+Address and IP+Offset, we perform prefetching with Bingo only for IP+Address event, and similarly only for IP+Offset event matching. We find that more than 99% of the performance improvement comes from the matching of IP+Offset event in Bingo. To observe the benefits of accurate event prefetching with IPCP, instead of training separately per IP, we train separately for each IP+Offset combination, i.e., (IP+Offset)CP. This significantly increases the conflicts in the IP table, however, it improves performance as well. On average, this improves performance by 0.61%. Figure 7 shows the performance improvement of performing IP+Offset-based training over IP-based training. We observe a performance improvement of as high as 18.3% for *mcf-1554B*. There is a 53.5% increase in the number of prefetch requests that the prefetcher issues, leading

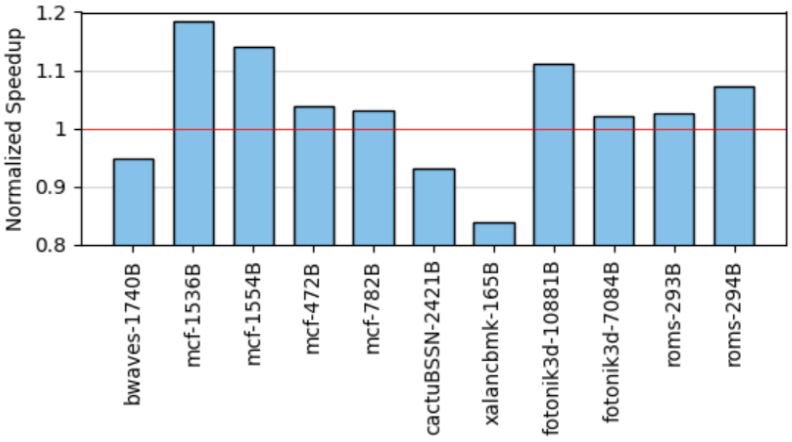

Figure 7: Performance improvement of (IP+Offset)CP over IPCP.

to pollution and hence, performance degradation, in some cases.

What works and what doesn't? The following summarizes the schemes that we experiment with, to improve performance with IPCP: (i) training for regions larger than 4 KB degrades overall performance as for most benchmarks, learning can be performed within a 4 KB region, (ii) a region-based prefetching component at L2 improves performance further by 1.8% on average, in the presence of IPCP at L1D as well as the state-of-the-art region-based prefetcher at L2, (iii) performing accurate training using IP+Offset instead of just IP, provides an overall performance improvement of 0.61%, and maximum of 18.3%; but it degrades performance significantly for some benchmarks, making the overall improvement conservative.

Future direction: The following describes the future research direction in this domain: Implementing an adaptive scheme that can figure out the optimal region size for training; the most performance improvement that we can gain from this is 3.5% on average across 46 memory-intensive SPEC CPU 2017 benchmarks.

4 Spatio-temporal Data Prefetching

The gains that a spatial data prefetcher can provide for irregular benchmarks are limited as it is unable to learn the access pattern from an irregular access stream. Combining the gains of a spatial and a temporal prefetcher can perform better than a stand-alone spatial or temporal prefetcher. With this goal in mind, we experiment with including the temporal prefetcher Irregular Stream Buffer (ISB) [11] as a new class in IPCP which is the state-of-the-art spatial prefetcher. Figure 8 shows the performance of state-of-the-art data prefetchers across 24 irregular SPEC CPU 2006 benchmarks. Here, we see a maximum performance improvement of 12.5% with PPF (Perceptron-based Prefetch Filter).

Spatial Temporal Memory Streaming (STeMS) [12] also exploits both spatial and temporal locality, however, they propose it in a synergistic manner. Whereas, moving forward, we perform spatio-temporal prefetching with independent spatial and temporal components,

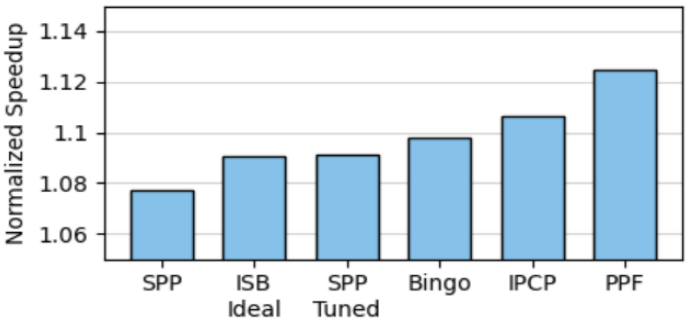

Figure 8: Performance improvement with data prefetchers across 24 irregular SPEC CPU 2006 benchmarks. We use a 48 KB version of Bingo to train at L1D. Apart from Bingo and IPCP, all prefetchers train at L2.

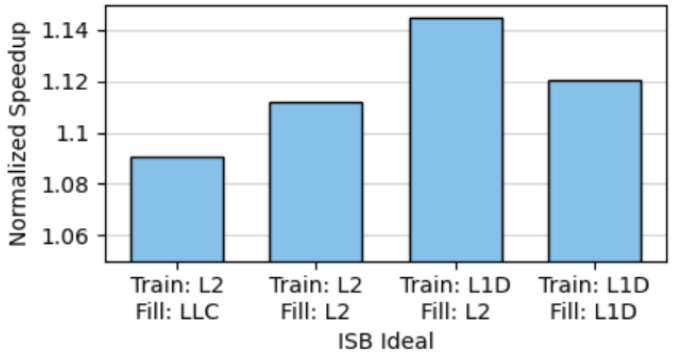

Figure 9: Performance improvement with ISB, varying the training and filling cache levels.

similar to how all the classes in IPCP are independent. Since, the scope of prefetching of a spatial prefetcher and a temporal prefetcher are orthogonal, we expect their gains to add up.

ISB ideal: We use an ideal version of ISB [11] which assumes no penalty or traffic for off-chip metadata accesses. Without these relaxations, ISB's performance improvement is lower. For the performance that Figure 8 shows, ISB trains with the L2 miss stream and prefetches till LLC. However, since IPCP highlights the benefits of prefetching at L1D, we experiment with ISB training at L1D as well. Figure 9 shows the performance improvement of ISB when we vary the training and filling cache levels. With training at L1D miss stream, and filling till L2, ISB ideal provides a performance improvement of 14.5%, which is 2% higher than the highest performing state-of-the-art data prefetcher i.e., PPF. Thus, we perform further experiments with this version of ISB.

IPCP++: On incorporating ISB as a new class in IPCP i.e., IPCP++, we observe an average performance improvement of 2.42% (Figure 10), and maximum performance improvement of 13.02% with *mcf-158B* (Figure 11). This shows that the scope of prefetching of IPCP (spatial prefetcher) and ISB (temporal prefetcher) is orthogonal to some extent. However, Figure 11 shows that while there is performance improvement with IPCP++ (v2) for most irregular benchmarks, omnetpp and xalancbmk show performance degradation with

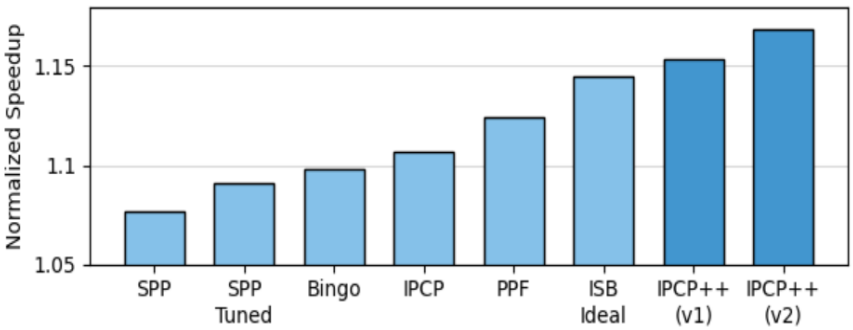

Figure 10: Performance improvement across 24 irregular SPEC CPU 2006 benchmarks, with IPCP++ (with ISB as a new class), v1: version 1, where ISB is the lowest priority class, v2: version 2, where ISB is the highest priority class.

IPCP++. This adverse effect is due to IPCP which degrades performance in omnetpp and provides relatively lower improvement in xalancbmk. Whereas, ISB ideal on its own performs better for these benchmarks.

Metadata utility in ISB: Prior works [11] [13] show that ISB incurs megabytes (MB) of metadata storage overhead. Thus, it uses off-chip memory to store the metadata and retrives it whenever required. However, [14] shows that this hampers performance in a multi-core environment, and so it proposes partitioning the shared LLC to store temporal prefetcher's metadata. Triage also uncovers an insight that the reuse of temporal prefetcher's metadata is significantly low. So, we study the storage requirements and reuse of metadata with ISB ideal for the 24 SPEC CPU 2006 benchmarks. We find that, on average, ISB requires 12.2 MB of metadata, and maximum 133.6 MB for *mcf-250B*. Further, we find that, on average, 45.9% of the entries in the physical to structual address mapping table of ISB, are never referenced; and 52.9% of the entries are referenced less than 25 times. Further, since ISB captures streams, we find that, for 84.2% of the streams that ISB captures, less than 10% of the stream is actually referenced. This shows a scope of improvement to extract important metadata, which we further discuss in future direction.

What works and what doesn't? The following summarizes the scheme to improve performance of irregular benchmarks with a new temporal prefetching class (ISB temporal prefetcher) in IPCP: for 24 irregular SPEC CPU 2006 benchmarks, including ISB as a new and high priority class in IPCP improves the overall performance by 2.42%. This scheme uses an idealized version of ISB which doesn't incur the penalty of storing metadata off-chip.

Future direction: The following describes the future research direction in this domain: (i) There is a scope to enable or disable either of the spatial or temporal component of the prefetcher depending on its usefulness (example, in omnetpp), to further improve performance; in an ideal scenario, this could provide up to 19.31% performance improvement, and (ii) There is a scope to extract useful metadata entries of the temporal prefetcher and discard the remaining entries for storage efficiency; [14] uses a modified version of Hawkeye [15] replacement policy for achieving this, however it still dedicates 512 KB or 1 MB of LLC storage to the prefetcher's metadata; a more dynamic partitioning could be useful to avoid thrashing

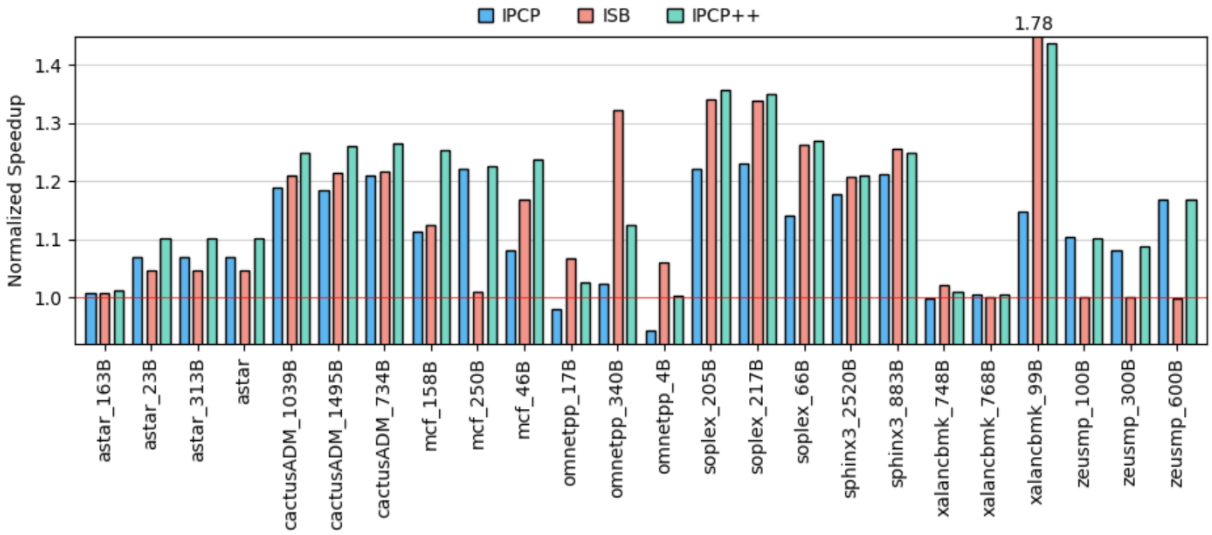

Figure 11: Performance improvement across 24 irregular SPEC CPU 2006 benchmarks, with IPCP, ISB, and IPCP++ (with ISB as a new class).

in the LLC, especially considering that LLC holds instruction, data, and translation blocks as well.

5 Conclusion

We uncover some of the limitations of IPCP, and show several experiment results that aim to eliminate those limitations. Following is a summary of the experiments that lead to performance improvement. IPCP makes 7.7% late prefetch requests on average, to rectify this, we increase the aggressiveness of the prefetcher, allow cross-page prefetching, and perform piggyback prefetching, and achieve an overall improvement of 13.61% across 46 memory-intensive SPEC CPU 2017 benchmarks. To improve the performance of IPCP further, we introduce a region-based prefetching component at L2, along with state-of-the-art region-based data prefetcher SPP, which leads to an overall performance improvement of 1.8% across 46 memory-intensive SPEC CPU 2017 benchmarks. To improve the performance of IPCP for applications with irregular memory access patterns, we include a new temporal prefetching class, ISB temporal prefetcher, in IPCP i.e., IPCP++, which leads to an overall performance improvement of 2.42% across 24 irregular SPEC CPU 2006 benchmarks.

References

[1] Pakalapati and Panda, "Bouquet of instruction pointers: Instruction pointer classifier-based spatial hardware prefetching," in *Proceedings of the 47th ACM/IEEE Annual International Symposium on Computer Architecture*, 2020.

[2] M. Bakhshalipour, M. Shakerinava, P. Lotfi-Kamran, and H. Sarbazi-Azad, "Bingo spatial data prefetcher," in *Proceedings of the 25th International Symposium on High-Performance Computer Architecture (HPCA)*, 2019.

[3] J. Kim, S. H. Pugsley, P. V. Gratz, A. L. N. Reddy, C. Wilkerson, and Z. Chishti, "Path confidence based lookahead prefetching," in *Proceedings the 49th Annual International Symposium on Microarchitecture MICRO 49*, 2016.

[4] E. Bhatia, G. Chacon, S. Pugsley, E. Teran, P. V. Gratz, and D. A. Jiménez, "Perceptron-based prefetch filtering," in *Proceedings of the 46th Annual International Symposium on Computer Architecture (ISCA)*, 2019.

[5] "Champsim simulator."

[6] T. Nakamura, T. Koizumi, Y. Degawa, H. Irie, and S. Sakai, "T-skid: Timing skid prefetcher," in *3rd Data Prefetching Championship*, 2019.

[7] A. V. Nori, J. Gaur, S. Rai, S. Subramoney, and H. Wang, "Criticality aware tiered cache hierarchy:a fundamental relook at multi-level cache hierarchies," in *Proceedings of the 45th Annual International Symposium on Computer Architecture (ISCA)*, 2018.

[8] V. Gupta, N. S. Kalani, and B. Panda, "Run-jump-run: Bouquet of instruction pointer jumpers for high performance instruction prefetching," in *First Instruction Prefetching Championship (IPC1) co-located with ISCA*, 2020.

[9] S. Srinath, O. Mutlu, H. Kim, and Y. N. Patt, "Feedback directed prefetching: Improving the performance and bandwidth-efficiency of hardware prefetchers," in *IEEE 13th International Symposium on High Performance Computer Architecture*, 2007.

[10] A. Ros and A. Jimborean, "Entangling prefetcher for instructions," in *IEEE Computer Architecture Letters*, 2020.

[11] A. Jain and C. Lin, "Linearizing irregular memory accesses for improved correlated prefetching," in *The 46th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO-46, Davis, CA, USA, December 7-11, 2013*, 2013.

[12] S. Somogyi, T. F. Wenisch, A. Ailamaki, and B. Falsafi, "Spatio-temporal memory streaming," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, 2009.

[13] H. Wu, K. Nathella, D. Sunwoo, A. Jain, and C. Lin, "Efficient metadata management for irregular data prefetching," in *Proceedings of the 46th International Symposium on Computer Architecture, ISCA 2019, Phoenix, AZ, USA, June 22-26, 2019*, 2019.

[14] H. Wu, K. Nathella, J. Pusdesris, D. Sunwoo, A. Jain, and C. Lin, "Temporal prefetching without the off-chip metadata," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2019, Columbus, OH, USA, October 12-16, 2019.*, 2019.

[15] A. Jain and C. Lin, "Back to the future: Leveraging belady's algorithm for improved cache replacement," in *43rd ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2016, Seoul, South Korea, June 18-22, 2016*, 2016.