RCTP : Region Correlated Temporal Prefetcher

Dennis Antony Varkey
Dept. of Computer Science & Engg.
Indian Institute of Technology Madras
Tamil Nadu, India
dennis@cse.iitm.ac.in

Biswabandan Panda
Dept. of Computer Science & Engg.
Indian Institute of Technology Kanpur
Uttar Pradesh, India
biswap@cse.iitk.ac.in

Madhu Mutyam
Dept. of Computer Science & Engg.
Indian Institute of Technology Madras
Tamil Nadu, India
madhu@cse.iitm.ac.in

***Abstract*—Hardware prefetcher is an essential component of modern processors that helps in boosting system performance by fetching the data before processor demands for the same. Hardware prefetching techniques have been proposed to exploit various kinds of access patterns. However, there are applications that are highly irregular in nature that evolved in the past decade, and have massive memory footprint. Temporal prefetching techniques are effective in predicting the future addresses of these irregular applications. Prior works on temporal prefetching use large data structures to store temporal patterns and future memory accesses are predicted using these patterns. However, these techniques predict future accesses only when there is a pattern that is already trained for a given cache line address.**

To address this issue, we propose Region Correlated Temporal Prefetcher (RCTP). Our technique correlates temporal patterns of memory regions and predicts future accesses for the region whose patterns are yet to be populated. Thus, RCTP helps in predicting cache line addresses whose first access is yet to happen unlike the traditional temporal prefetchers. We evaluate RCTP on SPEC CPU 2006, CRONO, and PBBS benchmark suites. RCTP outperforms the state-of-the-art temporal prefetcher named ISB by 26%, and a recent delta prefetcher called VLDP by 6%. This improvement comes with a hardware overhead of 1KB over ISB; however, RCTP does not require off-chip storage, unlike other temporal prefetchers.

I. Introduction

Traditionally, prefetching techniques provide good performance gain if the cache accesses are predictable and regular. Prefetching techniques such as stride, stream, and best offset are well known regular prefetchers. Stride prefetcher generally prefetches the cache lines which are at a particular stride away from the requested cache line. Next-line prefetcher [1] is a specific stride prefetcher that prefetches the next cache line. Best Offset Prefetcher (BOP) [2] is an enhancement of next-line prefetcher that looks at the previous strides taken by the program to find the best offset from the offset list. Once the best offset is found, the offset is added to the current cache line address and the new address is requested for prefetching.

Access Map Pattern Matching [3] is a region based prefetching technique that uses bitmaps for each cache line in the region and tracks whether the cache lines are accessed or not. A matching logic is used to find patterns in the bitmap and using this logic the predicted addresses are prefetched. Spatial Memory Streaming (SMS) [4] also uses the concept of region. For each cache block of a region, a bitmap is used to store the spatial correlation of the cache blocks within a region. When an access to a region recurs, using bitmap the spatially correlated accesses in the region are prefetched. Though, these regular prefetchers are effective for regular access patterns, these techniques provide limited effectiveness for irregular applications.

Applications that contain pointer-chasing codes, indirect references such as A[B[i]], where A and B are arrays are some of the examples of irregular applications. These applications consist of irregular strides that may span across multiple OS pages. Irregular strides are the differences in consecutive cache line accesses in the memory address space that lack any fixed strides. Though there are prefetching techniques that target irregular applications, such as irregular stream buffer (ISB) [5], temporal memory streaming (TMS) [6], sampled temporal memory streaming (STMS) [7], spatio-temporal memory streaming (STEMS) [8], and variable length delta prefetcher (VLDP) to name a few, there is still a huge gap that can achieved in terms of performance (Please refer to Section III) for the details.

The primary reason for this huge gap is that the state-of-the-art prefetchers rely on recurrence of accesses to trigger prefetch requests, which restricts the effectiveness. We propose a technique that does not reply on recurrence of accesses. We use access patterns of a fixed size memory region to predict the future memory accesses of other memory regions, through a concept of *region correlation.* To the best of our knowledge, this is the first work that exploits region correlation for prefetching in irregular applications. Overall, we make the following contributions:
(i) We motivate for the need of better prefetching techniques for irregular applications (Section III). (ii) We propose, region correlation based temporal prefetcher (RCTP) that exploits the behavior of one memory region to prefetch addresses for other memory regions (Sections IV and V). (iii) We evaluate RCTP across different irregular applications and show its effectiveness in terms of performance. RCTP outperforms ISB by 26% and VLDP by 6% (Section VI).

II. Related Work on Irregular Prefetching

Temporal streaming prefetchers are shown to be effective for irregular applications [5]. Hence for irregular memory access pattern we use a baseline *temporal streaming prefetcher*. A temporal streaming prefetcher stores *temporal streams* (patterns) of an application to predict its future accesses. A temporal stream is a sequence of cache line aligned addresses

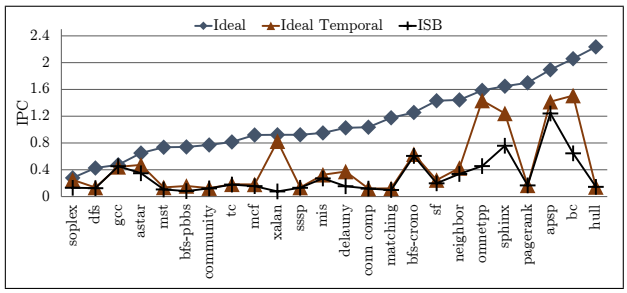

Fig. 1: IPCs of different prefetchers.

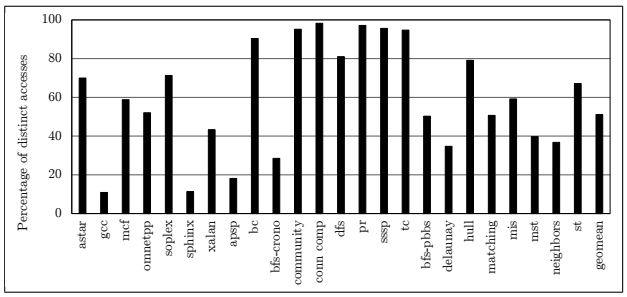

Fig. 2: Percentage of distinct accesses.

that are arranged in a chronological order of accesses. These streams recur temporally and hence can be used to predict the addresses of irregular applications. Temporal correlation based prefetchers such as Temporal Memory Streaming (TMS) [6] and Sampled Temporal Memory Streaming (STMS) [7] store memory accesses in the chronological order. When a memory access recurs, it finds the cache line accesses following itself and prefetches the upcoming addresses.

Jain and Lin have proposed Irregular Stream Buffer (ISB) [5] that uses program counter (PC) localization and address correlation while storing the meta-data of memory accesses. ISB introduced two address mapping caches (PS-AMC & SP-AMC) to store the meta-data. These new data structures arrange the meta-data of temporally correlated accesses in spatial order. SP-AMC stores the meta-data of PC localized temporal streams and the spatially aligned storage of meta-data helps in efficient reconstruction of the address stream. PS-AMC provides address mapping to SP-AMC. In ISB, when a cache line is accessed, the PS-AMC is looked for the address. On a successful match it points to a PC-localized stream in SP-AMC and the upcoming addresses are predicted. STeMS [8] uses both the spatial correlation as well as temporal correlation to predict the irregular access pattern.

VLDP [9] is a recent prefetcher that tracks complex address patterns. It uses a multilevel delta history to find a more accurate match for the future accesses. Another recent work called Signature Path Prefetcher (SPP) [10] also maintains a signature for each page. These signatures are made using the delta history. Using these signatures the upcoming accesses are predicted based on the path confidence.

III. The Problem

In this section we discuss the problem that we find in the state-of-the-art prefetching techniques for irregular memory accesses. Current temporal prefetching techniques do not predict the future memory accesses when a cache line is accessed for the first time or when the cache line has been accessed but the temporal pattern for the cache line is not available with the prefetcher as the pattern got evicted. In order to analyze and understand the implications of this design issue, we run experiments across three system configurations: ideal, ideal temporal and the state-of-the-art ISB.

- Ideal: This configuration mimics a system that shows the characteristics of a perfect *last-level cache* (LLC). A perfect LLC is a system configuration where all the memory accesses get hit in the cache (LLC or higher levels of cache) and the processor does not stall for DRAM access. In this configuration, the LLC-DRAM memory barrier is bridged. This helps us to gauge the opportunity in terms of IPC improvement when a perfect LLC prefetcher is implemented. A perfect prefetcher brings all the cache line addresses to the LLC before processor requests for the same. We implemented this model by setting the LLC-DRAM round-trip latency as *zero* cycles.
- Ideal temporal: This configuration mimics a system where a perfect temporal prefetcher is implemented. A perfect temporal prefetcher eliminates all the conflict and capacity misses. However, this prefetcher cannot eliminate cold misses as temporal prefetcher prefetches cache lines when they recur. This configuration helps us to know the possible IPC the best temporal prefetcher can achieve. We implement this by setting up a large LLC which would incur only cold misses.
- ISB: This configuration is implemented with the state-of-the-art temporal prefetcher [5].

Fig. 1 shows the difference in performance in terms of raw instructions per cycle (IPC) for these three configurations. We use irregular applications from benchmarks such as SPEC CPU 2006 [11], CRONO [12], and PBBS [13]. The benchmarks are sorted as per their IPCs for the ideal configuration and positioned from left to right. We see that there is a huge difference in IPCs for different configurations. For example, when the ideal temporal prefetcher is compared to the ideal configuration, difference in IPCs goes up to 2 with an average difference of 0.68. *This shows that state-of-the-art temporal prefetching techniques are not sufficient enough to hide the memory latency of irregular applications.* For SPEC CPU 2006 benchmarks that have the repetitive and predictable access patterns, using a perfect prefetching technique brings down the average IPC difference to 0.17. *However, in the case of CRONO and PBBS benchmarks, even an ideal temporal prefetcher does not contribute significantly to hide the memory latency and when the ideal temporal is compared with ISB, we find that ISB closely follows the ideal temporal. It learns the temporal patterns and prefetches efficiently.* The average IPC difference between these two configurations is very low (0.20). However, the IPC difference in the SPEC CPU 2006

benchmarks used here are higher showing there is still scope of improvement over ISB while prefetching for SPEC CPU 2006 benchmarks.

In order to further understand the behavior of irregular benchmarks that leads to such performance gap we further analyze the behavior of the ISB prefetcher. As mentioned earlier, a temporal prefetcher prefetches data for an address when that address is referenced more than once. So whenever a cache line is accessed by the processor for the first time, the prefetcher stores memory access patterns in its meta-data and the prefetcher does not have a pattern match in its meta-data to prefetch. Hence the prefetcher does not prefetch. We track the number of cache accesses referenced by the prefetcher for the first time where the prefetcher does not prefetch.

Fig. 2 shows the percentage of distinct accesses to the LLC compared to the total number of accesses. We can see that 40% of the total cache lines accesses are distinct (accessed for first time). In case of SPEC CPU 2006 benchmarks such as *astar, mcf, omnetpp, soplex*, and *xalancbmk*, the ratio of first cache block reference to the total number of cache block references is more than 40%. We also see in Fig. 1 that in the above mentioned five benchmarks, there is a significant gap between ideal temporal and ISB. In case of *sphinx, gcc*, and *libquantum*, ISB tries to bridge some gap as there are lot of repetitions. In the case of CRONO benchmark suite, seven out of nine benchmarks have more than 80% distinct accesses when compared to the total accesses. In all these cases the prefetcher does not prefetch. In the case of *bfs* and *apsp*, there is repetition of addresses and the gap between the ideal temporal and ISB is less. We also observe similar trends for PBBS benchmark suite in which five out of eight benchmarks have repetitions of less than 40%. Hence these benchmarks do not prefetch for a significant portion of their application run. *In conclusion, we find that in SPEC CPU 2006, CRONO, and PBBS benchmark suites, irregular memory accesses have very low repetitions, hence the number of prefetch requests that are generated by standard temporal prefetching techniques are less.*

Digging Deep: To understand the intricacies of temporal prefetcher that cause the gap when compared to the ideal temporal prefetcher, we analyze some of the graph benchmarks from CRONO, PBBS, and some irregular benchmarks SPEC CPU 2006. As mentioned earlier, the cache lines being accessed are highly distinct. To exemplify this behavior, if a graph has a small number of vertices or edges or the data set has less memory footprint it can fit in the meta-data (stored address patterns) of prefetcher . In subsequent accesses, the prefetcher can prefetch the memory accesses effectively using the stored temporal patterns.

Temporal prefetchers need a lot of hardware to store the meta-data. ISB needs 4 bytes to store each memory address. ISB also uses 8MB of off-chip space to store the patterns. ISB can store a graph, which has vertices or edges up to 2 million; or a data set that has a memory footprint of less than 2 million accesses. If a benchmark has more than 2 million distinct cache line accesses, it will swap in and out the meta-data from the off-chip storage, so the prefetcher will not find a matching temporal pattern that can predict the future accesses, even though the cache line accesses recur. The dependence of a prefetcher on its off-chip meta-data storage always remains a critical performance bottleneck. This also consumes DRAM bandwidth to swap in and out the meta-data. One of the possible solutions is to use a limited subset of temporal patterns to prefetch all the memory accesses. Being a subset, it requires less storage space. Also, DRAM bandwidth consumption will be reduced as the subset can always be on-chip. *In this paper, we propose a technique where we store a subset of meta-data and use it to predict future addresses.*

IV. Region Correlation

Temporal prefetchers use various correlation techniques like address correlation, PC correlation, etc., to infer temporal access patterns within memory streams. In this work, we propose a novel correlation technique called *region correlation*, that correlates temporal patterns across memory regions. A *region* is defined as a logical partition obtained by dividing the physical memory address space, as previously used in various prefetching techniques, such as SMS and AMPM, where a memory access pattern is bounded within a region. For a given memory access within a region, existing temporal patterns of a prefetcher may not predict future memory accesses whenever the number of accesses to that region is less, or when the meta-data for that pattern has been replaced due to limited hardware storage. For such cases, we propose to employ the notion of extrapolation of existing patterns from one region to other region. Among these patterns, we use the *region offset* of the current access to select applicable patterns that contain a memory access with the same offset, and the selected pattern can be used by the prefetcher to predict future memory accesses for this region. The region offset of a cache block is defined as the distance between the cache block and the first cache block of that region.

However, an offset match alone may select a temporal pattern that does not predict the actual memory accesses accurately. Though this method may improve the prefetch coverage and IPC of an application, it may also hamper the prefetch accuracy and DRAM bandwidth consumption. In fact, this method may degrade the performance of applications when accuracy falls below a certain threshold. *Hence, it is important to extrapolate and select a temporal pattern that closely matches the actual memory access stream.*

In order to prune the prefetch addresses generated by the temporal stream, we use *region-distance* based speculation. A region distance is the value obtained by dividing the difference in the addresses of two regions by the region size. We observe that correlations based on certain specific region distances may improve the prefetch accuracy, as compared to offset-based correlations alone. *Hence, in our proposed technique, we use region-distance based correlations on top of offset-based correlations.*

Fig. 3 illustrates three different scenarios where region correlation can be effective in predicting the future memory

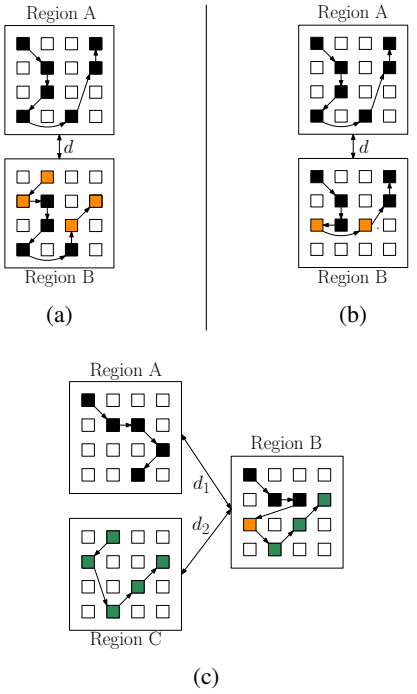

Fig. 3: Scenarios of region correlation. (a) single stream; (b) multiple partial streams; and (c) mixed streams.

accesses. Each region is represented as a set of cache blocks, which are aligned in a grid like structure. The cache blocks follow row-major ordering. A and C are the regions whose temporal patterns are used to generate prefetch requests for region B. The black arrows indicate the sequence of cache block accesses for a PC within a region. Each chain-like linked cache block represents a PC localized stream within a region. For each region, the shaded boxes indicate the cache blocks that have been accessed. The region-distance is indicated by d.

Based on our analysis of benchmarks, we find that there are three different scenarios in which stream(s) from different regions may correlate.

Scenario 1: Single stream. In Fig. 3a, we show a scenario where a partial access pattern of region A matches with the partial access pattern of B. So when we get a region offset match, we can prefetch using the temporal stream of A to partially predict the access pattern of B. In Fig. 3a, the black chain of accesses in region B is the common temporal pattern that both the regions have.

Scenario 2: Multiple partial streams. In Fig. 3b, we show the scenario where many partial access patterns of region B get a match with the access pattern of region A. This is a super-set of scenario 1. Fig. 3b shows two partial access patterns of region B that match with the access pattern of region A. The two black chains in the region indicate the overlap between region A and B and by using region correlation, we can prefetch the cache block accesses of region B.

Scenario 3: Mixed streams. Fig. 3c shows the scenario where multiple streams of different regions can be used to get a match in the region B. In this case, we show how the temporal streams of two regions A and C are used to find

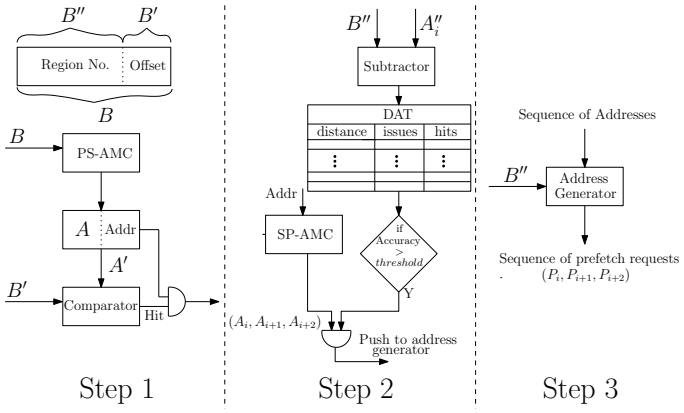

Fig. 4: Steps involved in RCTP.

the access pattern of region B. Fig. 3c shows a black colored access chain and a green colored access chain in region B that can be prefetched by region correlation. Black colored access chain shows the overlap of temporal pattern between region A and B, whereas the green colored chain indicates the overlap of temporal pattern between region C and B. There can also be a case where access patterns of two regions intersect on the same offset. All these cases can be taken care of by region correlation.

V. Region Correlated Temporal Prefetching

In this section, we discuss the proposed prefetching technique *Region Correlated Temporal Prefetching* (RCTP). Our technique generates prefetch requests in three steps: (i) temporal stream detection; (ii) region-distance based speculation; and (iii) address generation. Fig. 4 illustrates the steps involved in generating prefetch addresses using RCTP. It tries to predict future memory accesses on every LLC access and sends prefetch requests to DRAM.

In Fig. 4, B denotes the current address. A_i, A_{i+1}, A_{i+2},... are the addresses of temporal patterns stored in prefetcher. Finally, P_i, P_{i+1}, P_{i+2},... are the addresses predicted for prefetching.

Step 1: Temporal stream detection Given a cache line access, in this step, RCTP searches for a temporal stream for prefetching an address whose region-offset matches with that of the cache line access. Firstly, on a cache line access the PS-AMC is looked into to find the stream. An entry in PS-AMC contains a mapping to the structural address space, called *structural address*. PS-AMC being a 8-way cache, accommodates eight entries in a set. The region-offsets of these entries are matched with that of the current access. Among the matched entries, the least-recently-used entry is selected assuming that it will map to the temporal pattern closest to the current access pattern. The structural address of the selected entry is used to index into the SP-AMC. The index to SP-AMC is used to obtain the temporal stream (A_i, A_{i+1}, A_{i+2},...). This temporal stream is used next for region-distance based speculation.

Step 2: Region-distance based speculation In this step, RCTP prunes the addresses of the temporal streams based on

PS-AMC and SP-AMC	8K entries
prefetch$_{issues}$ and prefetch$_{hits}$	13 bits
distance	5 bits
distance accuracy table	17 entries
threshold (region-distance accuracy)	0.7

TABLE I: RCTP parameters.

region distance. As all the region correlations that generate addresses may not be accessed in future, we stop prefetching, if the region correlation between two regions is less accurate. We introduce a structure *distance accuracy table* (DAT) that stores the counters to calculate the accuracy of region-distances. The accuracy of a region-distance is the ratio of useful to total prefetched addresses that used the region distance. Useful addresses are the addresses that get cache hit DAT has three components: (i) region-distance, (ii) prefetch$_{issues}$ counter, and (iii) prefetch$_{hits}$ counter.

As shown in step 2 of Fig. 4, the region number of A_i is subtracted from that of B to find the region-distance. The region-distance is then looked up in the DAT. The region-distance is a key value to perform the look-up. On a successful look-up in the table, we obtain prefetch$_{issues}$ and prefetch$_{hits}$ for the region-distance. The accuracy is thus obtained as $\frac{prefetch_{hits}}{prefetch_{issues}}$ If the accuracy of the region-distance is above a *threshold* then the address of the temporal stream is passed to the address generator. When an address is issued for prefetching, the corresponding prefetch$_{issues}$ counter is incremented. On a hit to the address, the prefetch$_{hits}$ counter is incremented, thereby monitoring the accuracy of each region-distance.

Step 3: Address generation In this step, the address generator operates on the pruned temporal stream to generate the addresses for prefetching (P_i, P_{i+1}, P_{i+2}, ...). The stream is mapped to the current region during the process. The address generator takes addresses of the pruned temporal stream and splits each of them into two parts. As shown in Fig. 4, the most significant bits of an address indicate the region number and least significant bits indicate the region-offset denoted by A''_i and A'_i, respectively. In order to generate prefetch addresses, the region number of the current address B'' is appended to the region-offset of the addresses in the pruned temporal stream (A'_i, A'_{i+1}, A'_{i+2},...). The addresses (P_i, P_{i+1}, P_{i+2},...) are the probable candidates for prefetching and are generated in the following manner.

$$P_i = (B'' << region\text{-}size_{bits}) + A'_i$$
$$P_{i+1} = (B'' << region\text{-}size_{bits}) + A'_{i+1}$$
$$P_{i+2} = (B'' << region\text{-}size_{bits}) + A'_{i+2}$$

These addresses are then pushed to the prefetch queue.

Implementation details

Table I provides various parameters used in RCTP. The storage requirements per core are as follows:

- Distance accuracy table (17 entries)
  (5 + 13 + 13) bits $\times$ 17 = 65.86B
- PS-AMC (8K entries)
  (2B $\times$ 8K) = 16KB
- SP-AMC (8K entries)
  (2B $\times$ 8K) = 16KB
- Overall hardware (DAT + PS-AMC + SP-AMC)
  (65.86B + 16KB + 16KB) $\approx$ 33KB

Overall, RCTP has a hardware overhead of less than 33KB per core. Compared to ISB, RCTP has an additional on-chip hardware overhead of 1KB; there is no off-chip hardware overhead in RCTP, unlike as required by ISB (8MB).

For some benchmarks, region-distance accuracies vary over the period of execution. The region-distance accuracy may be lower in the initial stages, however it can improve over the period of execution. Hence for these benchmarks once a region-distance accuracy falls below the threshold, prefetches based on that region distance ceases for the entire run. To overcome this issue we reset the counters. The counters are reset after every fixed interval, called *reset interval*. We refresh the DAT entries after every 10K access to the LLC. The size of refresh interval is fixed based on the analysis explained in Section VI-C. We fix the size of DAT entries (prefetch$_{issues}$ & prefetch$_{hits}$) as 13 bits as it is sufficient enough for 10K cache accesses in a refresh interval. We consider region size to be 4KB. Using empirical evaluations, we find that this size performs the best.

Processor	1/4 out-of-order cores (4GHz, 8 Issue, 192 ROB size, x86 ISA)
L1 cache (I & D)	32KB/per core, private, 8-way, 2 cycle MSHR 4, 64B cache block
L2 cache	2MB/8MB, unified, 16-way MSHR 20/64, 64B cache block
DRAM	DDR3 1600 MHz 1/2 channels, 2 Ranks/channel, 8 Banks/Rank

TABLE II: Simulator configuration.

VI. Results

We use gem5 [14] simulator to evaluate RCTP on single-core and 4-core systems. Table II provides the configuration of the simulated system. We use benchmarks with irregular memory accesses. Table III lists the benchmarks used from SPEC CPU 2006, CRONO, and PBBS. We run the simulation for a representative 250M instructions. In case of SPEC CPU 2006 benchmarks, the simulations are fast-forwarded by 15 billion instructions. The number of instructions fast-forwarded for CRONO and PBBS depends on the loading of the graph and its creation.

We compare RCTP with two other techniques, ISB and VLDP. ISB is the state-of-the-art temporal prefetcher. VLDP is a recent prefetching technique that uses multiple levels of delta history pattern to predict future accesses. It achieves speedup where the address sequences have multiple repeating deltas. We choose VLDP over BOP [2] and SPP [10] as both the techniques work efficiently only for regular patterns. SPP does not perform well for benchmarks like *xalancbmk, mcf, astar, gcc* because of low spatial locality.

Suite	Benchmarks	Data-set
SPEC2006	astar, gcc, mcf, omnetpp, soplex, sphinx, xalancbmk	ref
CRONO	apsp, bc, bfs_crono, community, conn_comp, dfs, pagerank, sssp, triangle counting	California road network / synthetic
PBBS	bfs_pbbs, delaunay triangulation, hull, matching, mis, mst, neighbor, spanning forest	LiveJournal social network / synthetic

TABLE III: Benchmarks.

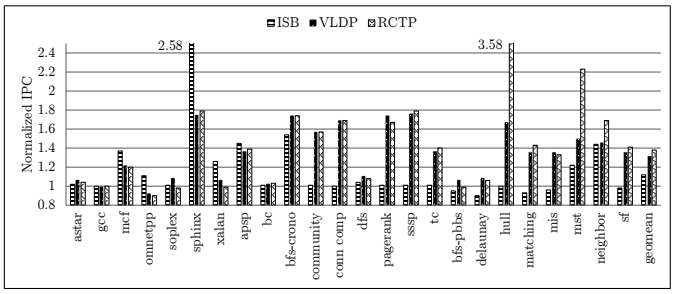

Fig. 5: Normalized IPC over no prefetching.

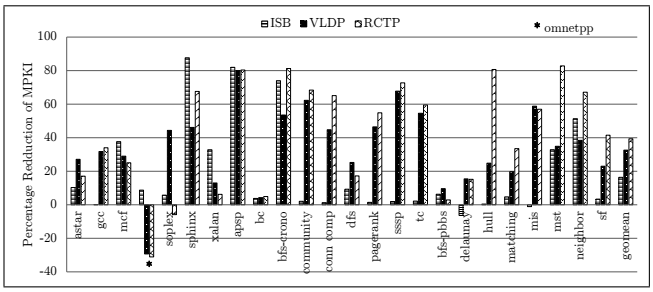

Fig. 6: MPKI reduction over no prefetching.

A. Single-core evaluations

We evaluate RCTP based on three metrics: (i) performance in terms of IPC; (ii) prefetch coverage; and (iii) prefetch accuracy. We use prefetch coverage as the reduction in MPKI of LLC, which is $(1 - \frac{MPKI_{technique}}{MPKI_{NOPF}})$, and prefetch accuracy $(\frac{prefetch_{hit}}{prefetch_{issued}})$ [15].

Fig. 5 shows the normalized IPC over no prefetching. Overall, on average, RCTP performs better than ISB and VLDP. The speedup of RCTP over no prefetching is 38%, whereas the speedups of ISB and VLDP are 12% and 32%, respectively. We observe a maximum speedup of 3.58x for benchmark *hull*. *Hull* being memory intensive as indicated in Fig. 1 achieves speedup as RCTP saves critical misses.

RCTP performs better than ISB for all the benchmarks of CRONO except *apsp*. For CRONO, most of the benchmarks have very low reuse of cache line addresses. However, for benchmarks, like *apsp* and *bfs*, the cache line reuse is high. Hence ISB improves the speedup. In case of *apsp*, ISB outperforms all other techniques. In case of *apsp* the difference in speedup between ISB and RCTP is 6%. RCTP matches or performs better than VLDP in all the benchmarks of CRONO except *pagerank*. For *pagerank*, RCTP performs better when region-based speculation is not used but in that case the prefetch accuracy is very low. If we use off-chip memory (as in the case of ISB to store temporal patterns at the DRAM) with RCTP, for benchmarks like *apsp* and *bfs*, it outperforms VLDP and ISB by 10% to 15%, respectively.

RCTP performs better than ISB and VLDP, with average (geomean) improvements of 55% and 23% in speedup, respectively, for the benchmarks of PBBS. Note that when compared to ISB, RCTP consistently performs better for all these benchmarks. This is due to low cache line reuse. For the benchmarks *bfs*, *delaunay*, and *mis*, VLDP performs slightly better with improvement of less than 5%.

For SPEC CPU 2006, ISB clearly outperforms both RCTP and VLDP. For *omnetpp*, there is a long gap between stream accesses. Hence the off-chip memory usage enables ISB to predict the pattern. Whereas in RCTP, inaccurate correlation leads to cache pollution thereby degrading performance. Fig. 6 shows that in the case of *omnetpp*, MPKI increases due to cache pollution. Similarly, for *xalancbmk* and *soplex*, there is low region-distance based correlation and hence the improvement in performance is also low. However, in the case of *sphinx* and *mcf*, region correlation helps to achieve marginal speedup. In case of *sphinx* and *mcf*, there are some region-distances that lie outside of what RCTP monitors. Using them would further increase the performance improvement. However, when off-chip memory is used with RCTP, it matches the performance achieved by ISB.

Fig. 6 shows the reduction of MPKI for different techniques. For the benchmarks *hull, mst, matching*, and *nearest neighbor*, RCTP improves performance and also has high coverage. This shows there is a direct correlation between performance improvement and coverage. For benchmarks like *bfs*, *community*, *sssp*, *tc* and *sf*, the coverage is slightly better as compared to ISB and VLDP, giving small performance improvement. We also see that in the benchmarks *mis* and *dfs*, VLDP has a better coverage giving a better IPC improvement. In *pagerank* we see that the coverage in the case of RCTP is more but VLDP prefetches more critical memory accesses, hence VLDP has slightly better IPC improvement when compared to RCTP.

In case of SPEC CPU 2006 benchmarks, the coverage of RCTP is lower than that of ISB. In benchmarks like *astar, mcf, gcc, and sphinx*, RCTP almost matches the coverage when compared to ISB, and hence it provides IPC improvement in these benchmarks. Whereas in *omnetpp* and *soplex*, RCTP issues useless prefetches that cause cache pollution, and hence we see that in these cases there is degradation of speedup with RCTP. If we use RCTP with off-chip memory then we see that most of the benchmarks give better coverage than ISB. In case of *omnetpp*, inaccurate region correlation causes cache pollution, thereby degrading the performance.

In Fig. 7, we plot the prefetch accuracy (in %) for different prefetching techniques. RCTP matches the prefetch accuracy of ISB and provides a better prefetch accuracy when compared to VLDP. In CRONO and PBBS, ISB has a high accuracy as

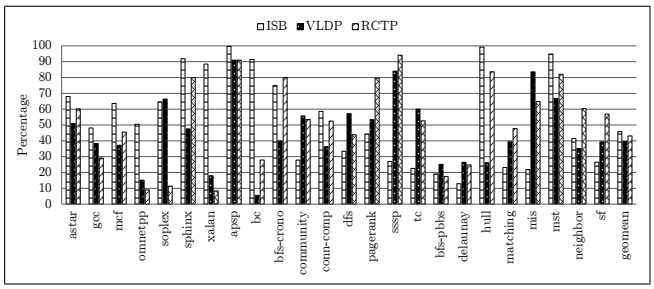

Fig. 7: Prefetch accuracy.

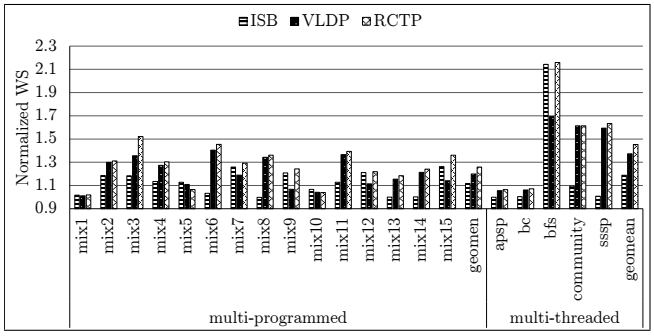

Fig. 8: Normalized weighted speedup for 4-core system.

the number of prefetches issued is very less and ISB does not improve the performance even though the accuracy is high. In benchmarks like *apsp*, *bfs*, *pagerank*, *hull* and *mst*, RCTP has high prefetch accuracy. As stated, *xalancbmk, omnetpp* and *soplex* have low region-distance based correlation, thus the accuracy reduces significantly. Prefetch coverage reflects the misses reduced by the prefetcher as well as any cache pollution induced by the prefetcher. If there is cache pollution, MPKI for that cache also increases. We see that RCTP reduces MPKI by 37.5%, whereas ISB and VLDP reduces MPKI by 16% and 32%, respectively.

B. Multi-core evaluations

We evaluate the multicore results using the weighted speedup metric [16], which is $\sum_{i=0}^{N-1} \frac{IPC_i{}^{together}}{IPC_i{}^{alone}}$. $\text{IPC}_i^{together}$ is the IPC of an application when simulated on an N-core system with other N-1 applications and IPC_i^{alone} is the IPC of that application run alone on the N-core system. The weighted speedup is normalized to no prefetching technique and has been plotted in Fig. 8.

The mixes of multiprogrammmed workloads are randomly generated using SPEC CPU 2006 benchmarks. These include both regular and irregular benchmarks. For multithreaded evaluations we use benchmarks from CRONO, where each benchmark spawns one thread for each core.

For multiprogrammed workloads, RCTP achieves 25.9% speedup over no prefetching. It outperforms ISB and VLDP by 14.17% and 5.99%, respectively. For multithreaded benchmarks, RCTP achieves 45.34% over no prefetching and performs better than ISB and VLDP by 26.51% and 8.07% respectively. Except for mix5 and mix10 of multiprogrammed

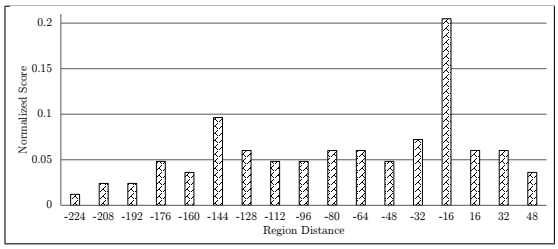

Fig. 9: Normalized score distribution of various region-distances.

workloads, for all other mixes RCTP performs better than ISB and VLDP. For mix5 and mix10, both VLDP and ISB perform better than RCTP. The reduction in accuracy in RCTP for mix5 and mix10 causes performance degradation. For other mixes RCTP has better accuracy.

The multithreaded workloads having low repetitions of accesses leads to low coverage in ISB. This causes reduced performance for ISB. In the case of VLDP, the inaccuracy in prefetching causes reduced performance. Overall we see that for multithreaded and multiprogrammed workloads, RCTP achieves better speedup when compared to ISB and VLDP.

C. Sensitivity analysis

In this section, we discuss the impact of different tuning parameters on RCTP. The parameters are (i) DAT entries; (ii) threshold of region-distance accuracy; and (iii) reset interval.

DAT entries: We run the RCTP simulation without region-distance speculation and store the prefetch accuracy of each region-distance. Once the simulations are over, we analyze the accuracy of each region-distance. Each region-distance is assigned a score of 1, 0, or -1 based on the accuracy. If the accuracy of a region-distance is greater than 0.7 (threshold of region distance accuracy) for at least 10,000 references during correlation, a score of 1 is given. Similarly, if the accuracy is less than 0.7 for more than 10000 references, a score of -1 is given. In any other case, a score of 0 is given. A score of 1 indicates a useful region-distance, 0 indicates neutral, and -1 indicates a useless region-distance for our technique. For each region distance, its score across all the benchmarks is summed up to obtain its cumulative score. All the region-distances that have a cumulative score of less than zero are omitted. The cumulative scores of the remaining region-distances are normalized and plotted in Fig. 9. The plot shows the normalized score distribution of region-distance across all benchmarks. The distribution shows the effectiveness of each region-distance. In the plot, we see the region-distance -16 is useful in most benchmarks. We see from the plot that all the 17 region-distances are effective irrespective of the benchmarks. Hence we choose to monitor these 17 region distances.

Threshold of region-distance accuracy: We analyze different thresholds of region-distance accuracy to find the optimal threshold. For this purpose, simulations are run on different threshold values keeping all other tuning parameters constant. Fig. 10 shows the sensitivity of RCTP to different threshold

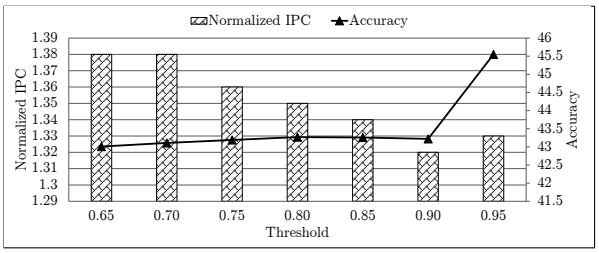

Fig. 10: Normalized IPC and prefetch accuracy over different thresholds.

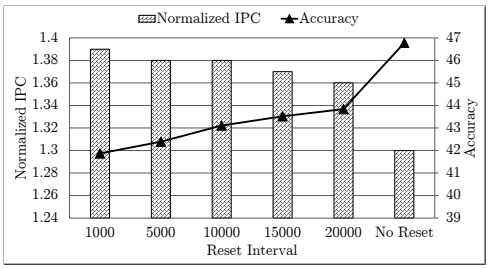

Fig. 11: Normalized IPC and prefetch accuracy over different reset intervals.

values; a bar is the geomean of the normalized IPC for a threshold value across all benchmarks. Similarly a point in the line graph is the geomean of accuracy across all benchmarks. The values in the figure show the desirable trend. As we increase the threshold value the normalized IPC decreases. This is due to reduced prefetching opportunities as prefetching stops at higher threshold; whereas the accuracy increases with the threshold value. We see that the normalized IPC values tend to get stagnant after 0.7. Similarly the accuracy decreases very gradually as we lower the threshold. Hence in RCTP, we use 0.7 as the threshold of region-distance accuracy.

Reset interval: We analyze and show the sensitivity analysis of reset intervals. In this regard, we plot the geomean of normalized IPC and prefetch accuracy for different reset intervals (1K, 5K, 10K, 15K & 20K access to LLC) along with a special case where we do not reset the DAT counters (Fig. 11). As the reset interval size increases, the normalized IPC decreases, whereas the prefetch accuracy increases. We restrict the prefetching once the region-distance accuracy falls below the threshold. As the size of the reset interval increases the restriction is induced for a longer interval, hence reducing the normalized IPC. The region-distance that has lower accuracy will get the opportunity to prefetch again after resetting the counters. As the size of reset interval increases the resetting of DAT counters decreases, hence the accuracy increases. In cases where we do not reset the DAT counters, we see that the accuracy is higher compared to cases where we reset the DAT counters. For some benchmarks such as *nearest neighbor* and *mst*, the region-distance accuracy is initially low. Hence, without reset intervals they stop prefetching in the early stages. This results in lower normalized IPC and higher accuracy.

We see that the change of accuracy is less than 1% between 20K interval and 10K interval. The normalized IPC also reduces when we increase the reset interval size from 10K to 15K. Normalized IPC at 5K and 10K are similar but 10K has better accuracy. Hence we choose 10K as we do not compromise on accuracy yet have speedup.

VII. Conclusions

In this paper, we introduced RCTP, a temporal prefetcher that uses temporal patterns of a memory region to prefetch for another memory region. Existing temporal prefetchers do not prefetch for the accesses whose temporal pattern is unknown to prefetcher. However, RCTP performed region correlation to predict the temporal pattern for accesses unknown to prefetcher. Thus LLC misses were avoided, improving the performance. RCTP outperformed the state-of-the-art temporal prefetcher, ISB by 26% and a recent delta prefetcher, VLDP by 6% for a single core system configuration. RCTP improves the performance for applications with irregular memory accesses, by using region correlation.

References

[1] A. J. Smith, "Cache memories," *ACM Comput. Surv.*, vol. 14, pp. 473–530, Sept. 1982.

[2] P. Michaud, "Best-offset hardware prefetching," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2016.

[3] Y. Ishii, M. Inaba, and K. Hiraki, "Access map pattern matching for high performance data cache prefetch," *Journal of Instruction-Level Parallelism*, vol. 13, pp. 1–24, 2011.

[4] S. Somogyi, T. Wenisch, A. Ailamaki, B. Falsafi, and A. Moshovos, "Spatial memory streaming," in *Computer Architecture, 2006. ISCA'06. 33rd International Symposium on*, pp. 252–263.

[5] A. Jain and C. Lin, "Linearizing irregular memory accesses for improved correlated prefetching," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 247–259, 2013.

[6] T. F. Wenisch, *Temporal Memory Streaming*. PhD thesis, Pittsburgh, PA, USA, 2007. AAI3289945.

[7] S. Somogyi, T. F. Wenisch, A. Ailamaki, and B. Falsafi, "Spatio-temporal memory streaming," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3, pp. 69–80, 2009.

[8] T. F. Wenisch, M. Ferdman, A. Ailamaki, B. Falsafi, and A. Moshovos, "Practical off-chip meta-data for temporal memory streaming," in *High Performance Computer Architecture, 2009.*

[9] M. Shevgoor, S. Koladiya, R. Balasubramonian, C. Wilkerson, S. H. Pugsley, and Z. Chishti, "Efficiently prefetching complex address patterns," in *Proceedings of the 48th International Symposium on Microarchitecture*, pp. 141–152, ACM, 2015.

[10] J. Kim, S. H. Pugsley, P. V. Gratz, A. N. Reddy, C. Wilkerson, and Z. Chishti, "Path confidence based lookahead prefetching," in *Microarchitecture (MICRO), 2016.*

[11] J. L. Henning, "SPEC CPU2006 benchmark descriptions," *ACM SIGARCH Computer Architecture News*, vol. 34, no. 4, pp. 1–17, 2006.

[12] M. Ahmad, F. Hijaz, Q. Shi, and O. Khan, "Crono: A benchmark suite for multithreaded graph algorithms executing on futuristic multicores," in *Workload Characterization (IISWC), 2015 IEEE International Symposium on*, pp. 44–55, IEEE, 2015.

[13] J. Shun, G. E. Blelloch, J. T. Fineman, P. B. Gibbons, A. Kyrola, H. V. Simhadri, and K. Tangwongsan, "Brief announcement: the problem based benchmark suite," in *Proceedings of the ACM symposium on Parallelism in algorithms and architectures*, pp. 68–70, ACM, 2012.

[14] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, *et al.*, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.

[15] S. Srinath, O. Mutlu, H. Kim, and Y. N. Patt, "Feedback directed prefetching: Improving the performance and bandwidth-efficiency of hardware prefetchers," in *High Performance Computer Architecture, 2007. HPCA 2007.*

[16] A. Snavely, D. M. Tullsen, and G. Voelker, "Symbiotic jobscheduling with priorities for a simultaneous multithreading processor," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 30, pp. 66–76, 2002.