Data Prefetching

CS 230 : Project

Atharva Tambat (210070014)
Adarsh Reddy Madur (210050091)
Anand Narasimhan (210051001)

Table of Contents

Introduction

The project aims to try and better the `icpc` prefetcher, that won the 3rd DPC (Data Prefetching Championship), for various workloads such as SPEC, graph analytics, SAT solvers and servers.

As always, our aim is to reduce the memory bottleneck through some clever prefetching.

Some SPECulation

Our SPECulation...

- We expect the SPEC to comparatively have more well behaved memory patterns. That is, they **may** have memory patterns :)

Some SPECulation

Our SPECulation...

- We expect the SPEC to comparatively have more well behaved memory patterns. That is, they **may** have memory patterns :)
- Graph and SAT workloads, for the most cases, may not have well behaved memory access patterns. They have more random accesses, making it harder to SPECulate what to prefetch.

Some SPECulation

Our SPECulation...

- We expect the SPEC to comparatively have more well behaved memory patterns. That is, they **may** have memory patterns :)
- Graph and SAT workloads, for the most cases, may not have well behaved memory access patterns. They have more random accesses, making it harder to SPECulate what to prefetch.
- Servers also, we think, might have random accesses because people (clients) access the server randomly. But once the request/command is sent by the client, server would be more well behaved in memory accesses (if single thread!). A muti-threaded, multi-cliented server would most definitely have random accesses.

Overview (10K view)

We explored the following possibilities for improving the prefetcher:

- **Complex Stride:** We observed (in `ipcp` paper) that the IPC was low when complex stride was being used the most. Naturally, one would think that improving complex stride would improve the IPC. Right??!! Atleast we thought so. But it turns out that ....
- **Page based:** In graph and SAT solvers, there is lot of jumping around in memory accesses. Many consecutive accesses will be in different pages. We explored the possibility of using pages to prefetch - to no avail :(
- **Miscellaneous:** Won't say more, lets dive in....

Improving Complex Stride Prefetching

Our observation - low IPC = high fraction of complex stride prefetches
So,

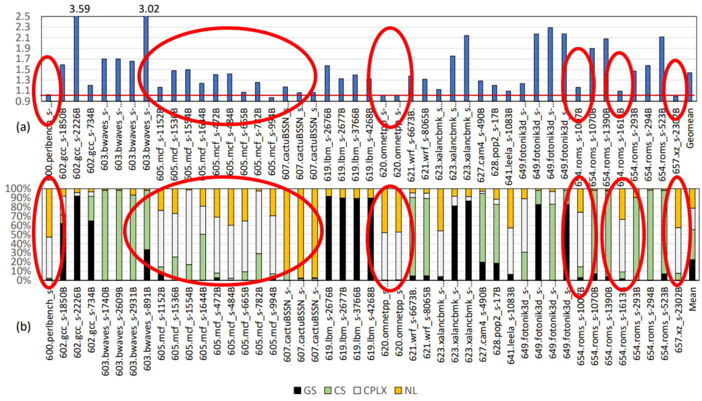


Figure: Low IPC where complex stride has high percentage

Improving Complex Stride Prefetching

- **Our hypothesis:** Reduce the number of miss-classified complex stride

Improving Complex Stride Prefetching

- **Our hypothesis:** Reduce the number of miss-classified complex stride
- **Our solution:** (Signature $\bigoplus$ stride) is not enough. Add more "features" - help correctly classify strides as complex

Improving Complex Stride Prefetching

- **Our hypothesis:** Reduce the number of miss-classified complex stride
- **Our solution:** (Signature $\bigoplus$ stride) is not enough. Add more "features" - help correctly classify strides as complex

- **Our implementation:** Add more "features" like
  1. PC $\bigoplus$ prev. PC $\bigoplus$ prev. prev. PC
  2. Page address $\bigoplus$ confidence

Improving Complex Stride Prefetching - Details

- Update confidence of a particular feature, if actual stride $=$ predicted stride
- If weighted sum of confidences of all features $\geq$ THRESHOLD $\rightarrow$ prefetch

Heuristics for the features

1. PC $\bigoplus$ prev. PC $\bigoplus$ prev. prev. PC - SPEC $+$ Graph Analytics $\uparrow$
2. Page address $\bigoplus$ confidence - Graph Analytics $\uparrow$ - “complex” pattern of page access

Improving Complex Stride Prefetching - Results

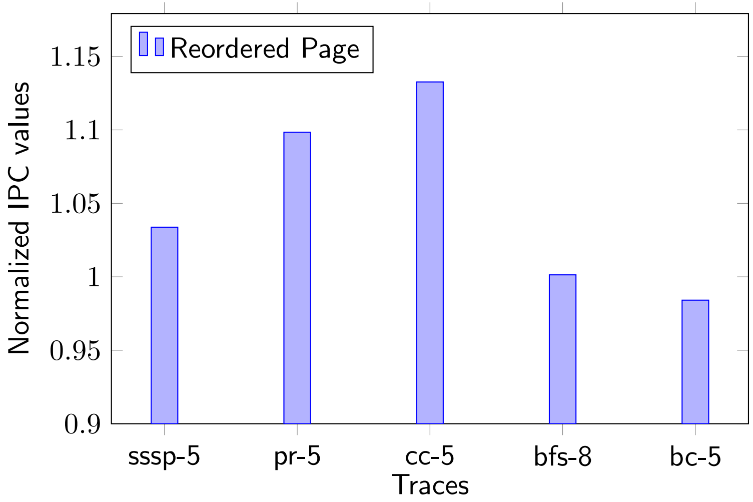


Figure: Normalized IPC for Graph workload

Improving Complex Stride Prefetching - Results

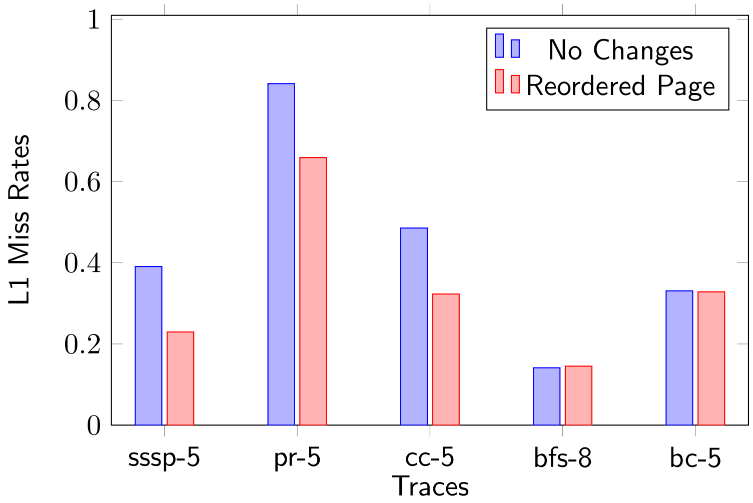
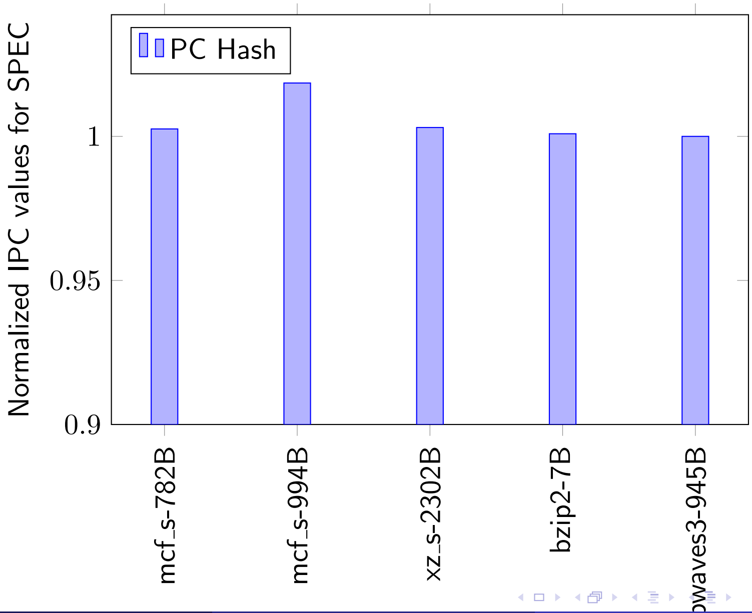
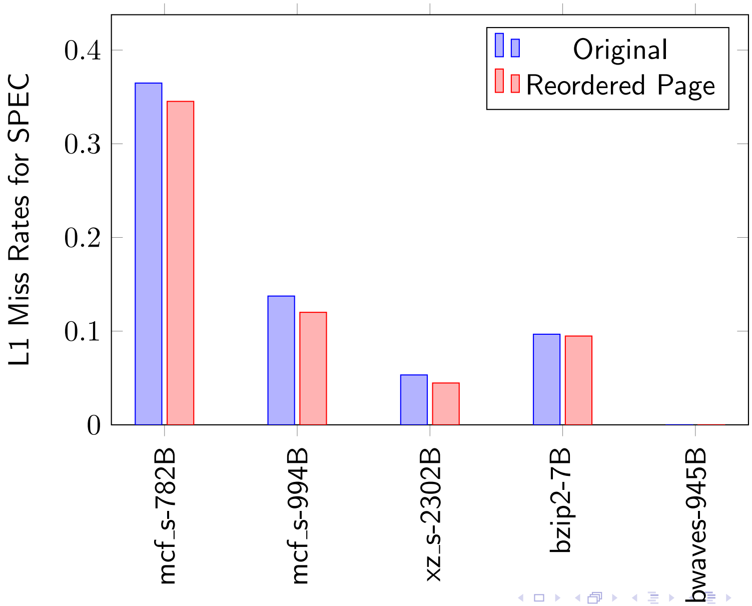


Figure: Miss Rate for Graph workload

Improving Complex Stride Prefetching - Results

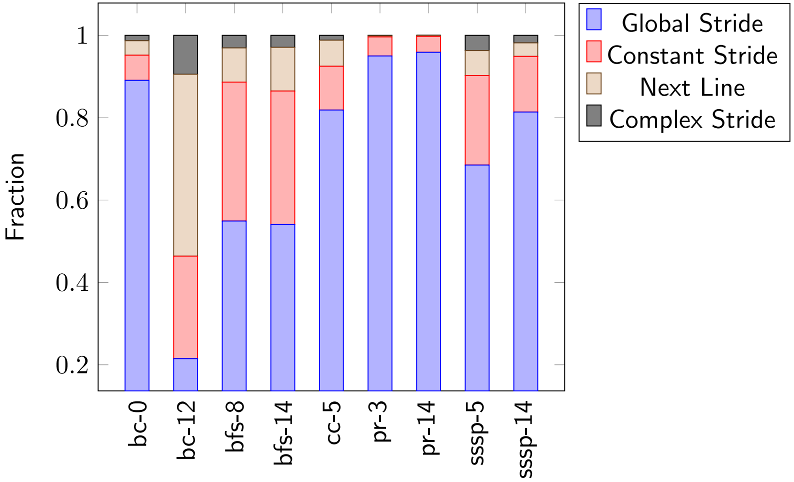


Improving Complex Stride Prefetching - Results



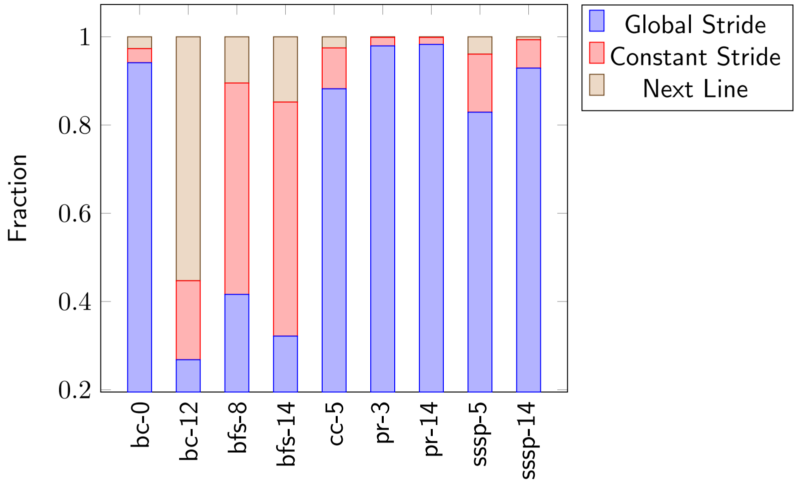
Original `ipcp` : L1

Fraction of L1 prefetches in each of the **prefetch classes** in the original `icpc` prefetcher, using **graph** traces.



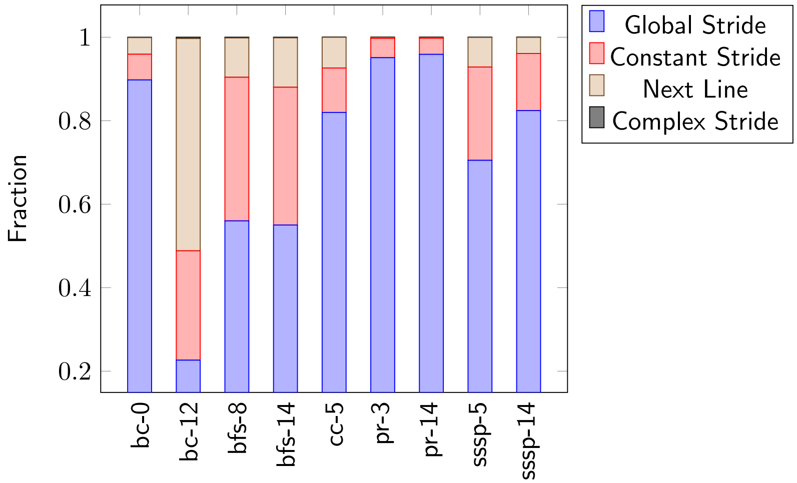
Original ipcp : L2

Fraction of L2 prefetches in each of the **prefetch classes** in the original `icpc` prefetcher, using **graph** traces.



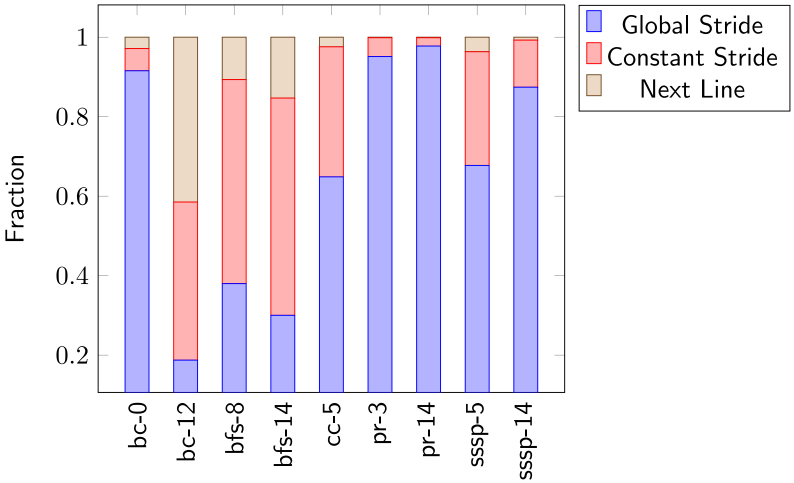
Modified ipcp (Added pc correlation) : L1

Fraction of L1 prefetches in each of the **prefetch classes** in the modified icpc prefetcher (Added pc correlation), using **graph** traces.



Modified ipcp (Added pc correlation) : L2

Fraction of L2 prefetches in each of the **prefetch classes** in the modified `icpc` prefetcher (Added `pc` correlation), using **graph** traces.



Improving Complex Stride Prefetching - Observations

- **Reordered Page:** refers to changing the order in which the types of strides are checked

- **Our Observation:** On printing out the number of prefetch requests for different stride types,
  1. # of strides classified as complex increases
  2. # of strides classified as GS (Global Stride) almost becomes 0

Improving Complex Stride Prefetching - Observations

- **Our conclusion:** Reordering of the prefetch classes is a really major factor. It changes the share of prefetches of each class by a lot. This is because the pattern that each of the classes identify is not completely independent of the other.
- They are correlated - the type of stride kept in lower preference, is maybe, not able to gain enough confidence on a prefetch address.

Pages

- All the memory that is used for a particular process/program may not have a continuous segment of memory allocated in the DRAM.
- OS deals with memory in **pages**. A page is the unit of memory brought for processing from the **off-chip** memory (harddisk, SSD etc).
- It may (most certainly) happen that a page brought from off-chip memory is stored at completely different locations of the DRAM, hence making the address space for a process discontinuous.
- Graph and SAT traces may have a lot of jumping around, across pages, due to random access patterns.

Page Prefetching

- Our idea is to prefetch an entire page if there are $N >$ THRESHOLD consecutive accesses in the same page.
- Each page has 64 lines, so we prefetch in batches so that we do not fill the prefetch queue.
- This prefetch takes up a lot of space, so, we prefetch it to the last level cache (LLC).
- Even by looks, one can guess that this won't work :) (Why did we try then? No answer, we just tried with HOPE). There is a lot of cache pollution due to this prefetch, hence the IPC decreases.

Some better methods....

Here are some more prefetching methods (we did NOT implement them)

- There are some better prefetching techniques that are better for random access patterns.
- These prefetchers, called temporal prefetchers, usually use off-chip memory to store the correlation data. But there is no way to access off-chip data in ChampSim (to the best of our knowledge). Example: ISB prefetcher
- **Triage:** This prefetcher does temporal prefetching without the off-chip memory accesses. We didn't try to implement it due to the lack of time (and the code we expect will be larger than `ipcp` code).

Removing Complex Stride from the Bouquet

- After all the added features, we went a step further and removed the complex stride classification entirely. This was very committal, worked very well for some traces, but did not work for a lot of other ones.
- A reason for this, similar to a reason for features, is broadly the ineffectiveness of the CPLX classification, a cause of that being the low confidence(0) for prefetching in CPLX. This led to some prefetches that weren't useful, and is the basis for the extra features that we introduced for the confidence as well.

CPLX in L2C

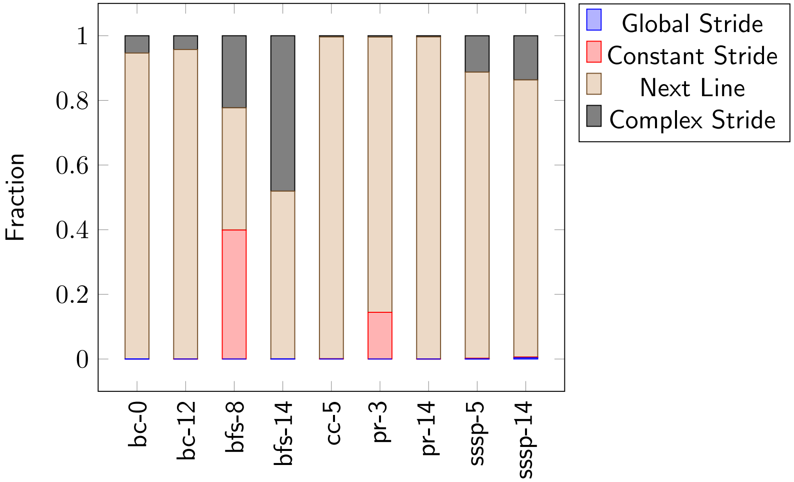
- Here, we decided to go in the opposite direction, and show why CPLX in the L2C wasn't a good idea. One reason is evident from the above discussion : a low threshold, and sending it to L2 with that threshold would not be a good idea, so we passed it to L2 only if the confidence was over a greater amount than usual. However, it still didn't perform as well.
- Biswa sir also explains it in another way (in one of his other papers), in terms of the degree of prefetching, which we did not play around with. Essentially, CPLX requires a lower degree of prefetching since the stride patterns are irregular, which is in contrast to CS and GS and so there is no need to put it in L2 as well.

Improving the Next Line Prefetcher

- The next-line prefetcher, as we observed in the graphs above, is accessed quite frequently. So we thought of making it better!
- The idea is similar to the bestoffset prefetcher.
- We learn the best offset by observing the last 1000 or so deltas (current address - previous address). The most frequent delta will be our current offset for prefetching.
- This method is giving a good improvement in most low performing graph based benchmarks when the prefetch classes are being reordered to give less prominence to GS and CS.

Reordered Prefetch Classes : L1

Fraction of L1 prefetches in each of the **prefetch classes** when the prefetch classes are reordered (GS < CS < CPLX), using **graph** traces.



Results

Here we have a code for each change that we made: (for making the table in the next page)

- **1** : Adding the pc based correlation to complex stride.
- **2** : Stopping the prefetches if MSHR is 3/4 filled.
- **3** : Modifying the next-line prefetcher in L1 cache.
- **4** : Implementing page based prefetching for LLC.
- **5** : Adding the page based correlation to **1**. That is pc + page based correlation to complex stride.
- **6** : Reordering **5** with priority, CPLX > CS > GS.
- **7** : Removing the CPLX prefetching.
- **8** : Reordering the next-line (**3**) modified file to have the priority CLPX > CS > GS.
- **9** : Implementing CPLX in L2 cache also.

Results

Here is the summary of IPC values relative to the IPC of original `ipcp`. Relative IPC > 1 is improvement and < 1 is decrease in performance. The traces are graph based.

Trace	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
bc-0	1.0015	1.0090	1.0537	0.9483	1.0015	1.0461	1.0015	1.1356	0.9996
bc-12	1.0020	1.0008	1.0867	0.9736	1.0019	1.0025	1.0020	1.0944	1.0002
bfs-8	1.0008	1.0010	0.9594	1.0010	1.0008	0.9802	1.0006	0.9792	1.0009
bfs-14	1.0016	1.0000	0.9192	1.0351	1.0026	0.9671	1.0016	0.9193	1.0012
cc-5	0.9770	1.0157	1.0035	0.8935	0.9777	1.1319	0.9819	1.2508	0.9980
pr-3	0.9620	0.9623	0.9620	0.8222	0.9621	1.0960	1.0000	1.2077	1.0000
pr-14	0.9999	1.0002	0.9999	0.8298	1.0000	1.1393	1.0000	1.2553	1.0000
sssp-5	0.9990	1.0071	1.0072	0.8391	0.9990	0.9674	0.9992	1.1563	1.0004
sssp-14	1.0002	1.0109	0.9993	0.9021	1.0009	1.0343	0.9992	1.2222	0.9999
GM	0.9937	1.0007	0.9979	0.9131	0.9940	1.0234	0.9985	1.1297	1.0000

Bye...

We enjoyed doing this project. We would have liked to explore many more things but due to the time crunch we couldn't. The project was definitely a roller-coaster and full of surprises. None or the intuitive changes improve the prefetcher. Well anyways, this means its the end of the course :(

CS 230 was a really fun course....

THANK YOU!