CS-305 Project

Team Zero Dawn

Introduction to IPCP

- IPCP classifies program counters into known prefetching ideas
- Namely - Global Stream, IP Stride, Complex Stride and Next Line

Reference: https://www.cse.iitb.ac.in/~biswa/IPCP_ISCA20.pdf

The Goal

To improve performance of IPCP where it is low ☺️

Where is it low?

In `perlbench`, `mcf`, `omnetpp`, `leela`, `xz`

Why is it low?

600.perlbench_s-570B

About the benchmark

The workload of this benchmark consists of:

SpamAssassin

(A mail spam filter)

MHonArc

(An email to html converter)

Specdiff from SPEC 2017

A utility tool that checks the Benchmark output

All of the loads have a thing in common which is …

String Processing

So we can expect large chunks of contiguous access patterns from the memory.

Why improvement is low?

- Perlbench is mostly insensitive to prefetching and the L1D hit-rate is already ~99.95% which is very high and there is little improvement in implementing a prefetcher.
- The working set size of this string processing application is small and the addition of the prefetching unit itself adds overhead to the runtime of the benchmark.
- Perlbench has a limited need for cache capacity and can be well executed without the need to regularly lookup the main memory.

Improvement Efforts

- Since the coverage by the global streaming class was the least for the perl benchmark, decreasing the size of the global history buffer slightly increased the IPC.
- Next line class has good coverage, so the priority of prefetching was changed by making the next line prefecture first, followed by GS, CS and CPLX classes, which gave some improvements.
- Decreasing the threshold value of confidence for CS prefetching also helped in slight improvement.
- Prefetch Degree updation based on prefetch hits/misses and confidence.

Some metrics

	Baseline	IPCP	IPCP-modified
IPC	1.38966	1.39974	1.40294
L1D miss rate	0.049%	0.033%	0.036%
L2C miss rate	72.35%	47.85%	29.95%
IPC Improvement	1.0000000000	1.00725357282	1.00955629435

We can observe a substantial improvement in the hit rate of the L2 cache which slightly increases the IPC despite the slight decrease in L1D hit-rate. Due to the very high rate of the baseline, the observed improvement is very less.

641.leela_s-1083B

About the benchmark

- This benchmark is a Go playing engine featuring Monte Carlo based position estimation

Why Less Improvement with IPCP?

- Leela have 98.8% hit rate in L1D and 96% hit rate in L2C with no prefetcher
- This testbench is not too much memory intense and have greater temporal locality
- Using IPCP reduced number of misses by nearly 42% but the number of misses are already low with no prefetcher, hence we were unable to find any substantial improvement

Some Improvements

- Since most of the IP's are being classified as CPLX, hence increasing the prefetch degree for CPLX class slightly improved the IPC
- Checking MSHR occupancy to decide the fill level of prefetching data and using a two bit ip_valid field which will increase for every memory access by that ip and decrease in case of conflict with another ip also slightly improved the IPC (to 0.763654)

Some Results

Prefetcher	No Prefetcher	IPCP	Modified with MSHR occupancy
IPC	0.753105	0.763613	0.763654

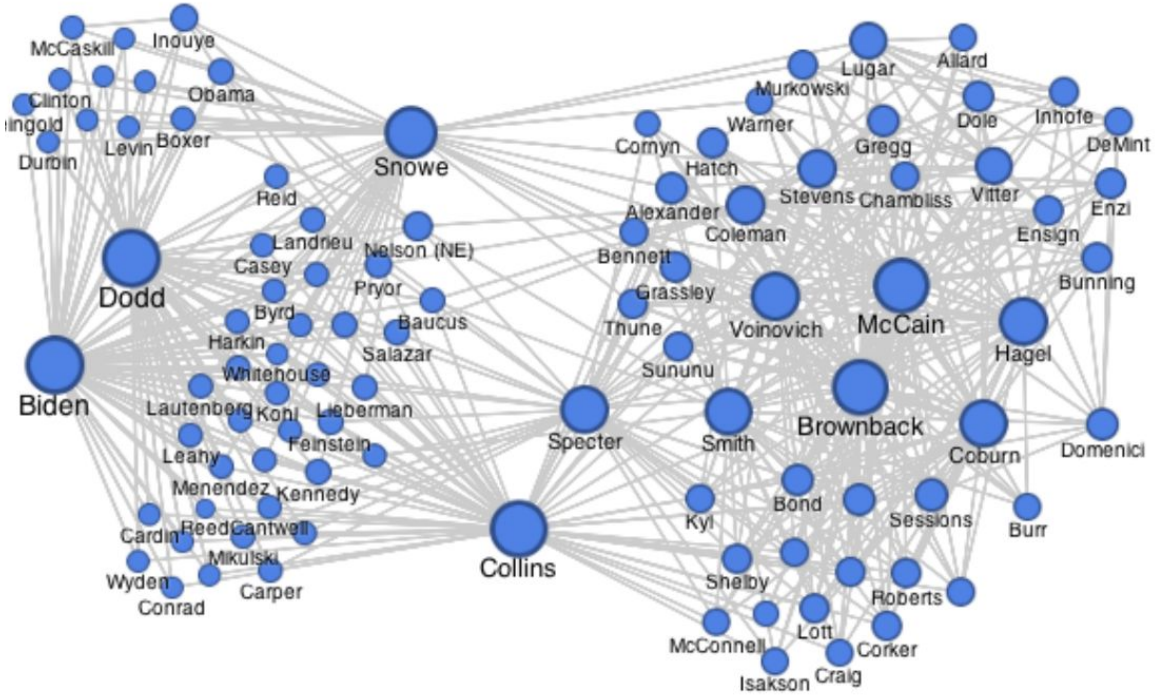
CPLX class Prefetch Degree	3	5	9	13
IPC	0.753105	0.763714	0.763855	0.763985

620.omnetpp_s-141B

About benchmark

Ethernet Simulation:

1) Physical layers
2) Traffic generation
3) Discrete event simulator



What to expect?

1) Random memory access are inherent. For eg : send a packet to randomly selected node in the network
2) Graphs traversals are involved. Eg : Packet traversal, shortest path algorithms.
3) Should we expect good locality access?
4) Can we expect 100% coverage? No.

Cannot predict random selection of nodes with any learning scheme.

Some metrics

	Baseline	IPCP
IPC	0.309583	0.32368
L1 cache hit rate	93.34%	93.56%
L2 cache hit rate	29.21%	40.49%
LLC cache hit rate	19.72%	24.91%

High L1D hit rate, locality is expected

Improvement mainly coming from L2, LLC

Performance improvement : 4.5%

*Above metrics are based on only loads

Why Low improvement

1) Low prefetching accuracy. Leading to cache pollution too

   Eg: L2 prefetches more lines than the no of request!!! (about 50%)

2) Probably CPLX class unable to capture random access and graph traversal algorithm's memory access
3) Maybe the program is already close to max theoretical speed. (Assuming random access cannot be predicted)

What's next line?

Our efforts : Further improvement

1) Disabling CPLX prefetching : Improved a further 1%
2) Adjust degree based on confidence in cplx : Improved further by 0.4%

   Advantage : Less cache pollution

   Tweaking existing parameters also didn't bring any further improvements beyond 1%

3) Default NL prefetching at L2 : Very little further improvement (0.27%)

605.mcf_s-994B

605.mcf_s-994B

- Benchmark derived from single-depot vehicle scheduling in public mass transportation.
- High miss rate in L2C and LLC already and IPCP degraded the performance.

Some metrics

	Baseline	IPCP
L1D hit rate	88.6%	89%
L2C hit rate	43.8%	41%
LLC hit rate	46.7%	37%

- The coverage of NL is very high and a lot of data is uncovered
- We can expect good locality here because of the L1D hit rate

Some experiments

- The baseline IPC for 10 + 10 instructions is 0.406337 and IPCP produced a value of 0.382881
- When the NL part of the prefetcher was removed, an improvement of 0.6% was observed over baseline
- When the NL threshold was decreased, an improvement of 0.4% was observed over baseline
- When prefetch control based on page numbers was added, an improvement of 5.5% was observed over IPCP but still 0.6% degradation over baseline

Observations

- Next line prefetching giving degradation may imply that there is some back-and-forth accesses in program
- As with the other benchmarks, the L1D hit rate here is already high (~89%), which makes the prefetching effort have little value

657.xz_s-2302B

About the benchmark

- xz Data Compression - entirely in memory
- No IO - focusing on performance of CPU, memory

Some metrics

	Baseline	IPCP
IPC	0.819729	0.811273
L1D cache hit rate	96.43%	96.54%
L2C hit rate	50.16%	47.26%

- This benchmark is similar to omnetpp and mcf - expect high locality as L1D hit rate is high

Attempts at Improving

Similar to what was done for the previous benchmarks.

- When NL prefetching is removed an improvement of 0.4% over IPCP is observed
- When NL threshold is decreased, an improvement of 0.1% over IPCP

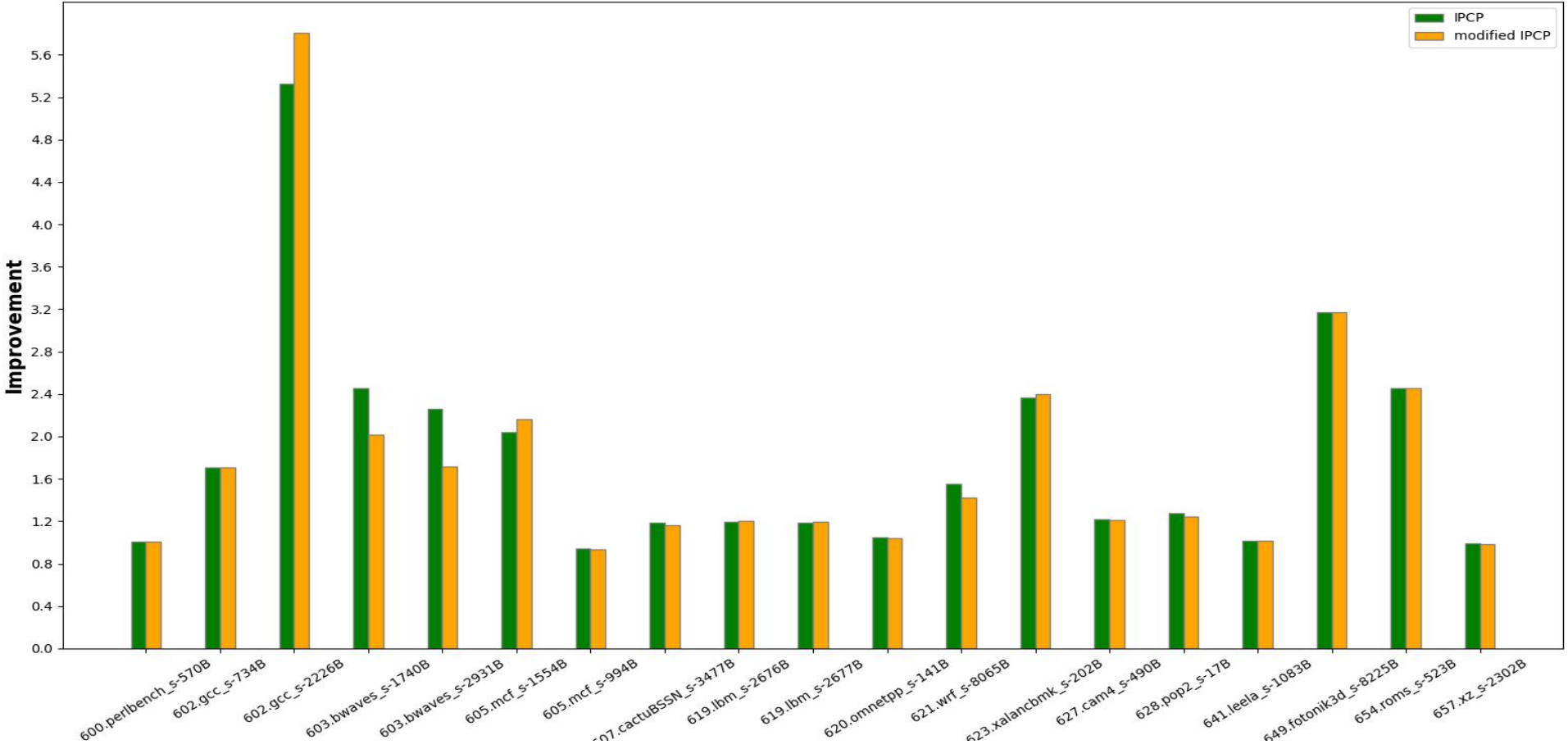
What could we do?

Ideas

Modification Idea to IPCP

Used MSHR occupancy in L1D cache to decide fill_level for the prefetch. Used a 2-bit ip_valid field which increases upon accessing the IP table by the same IP and decreases on being accessed by a different IP

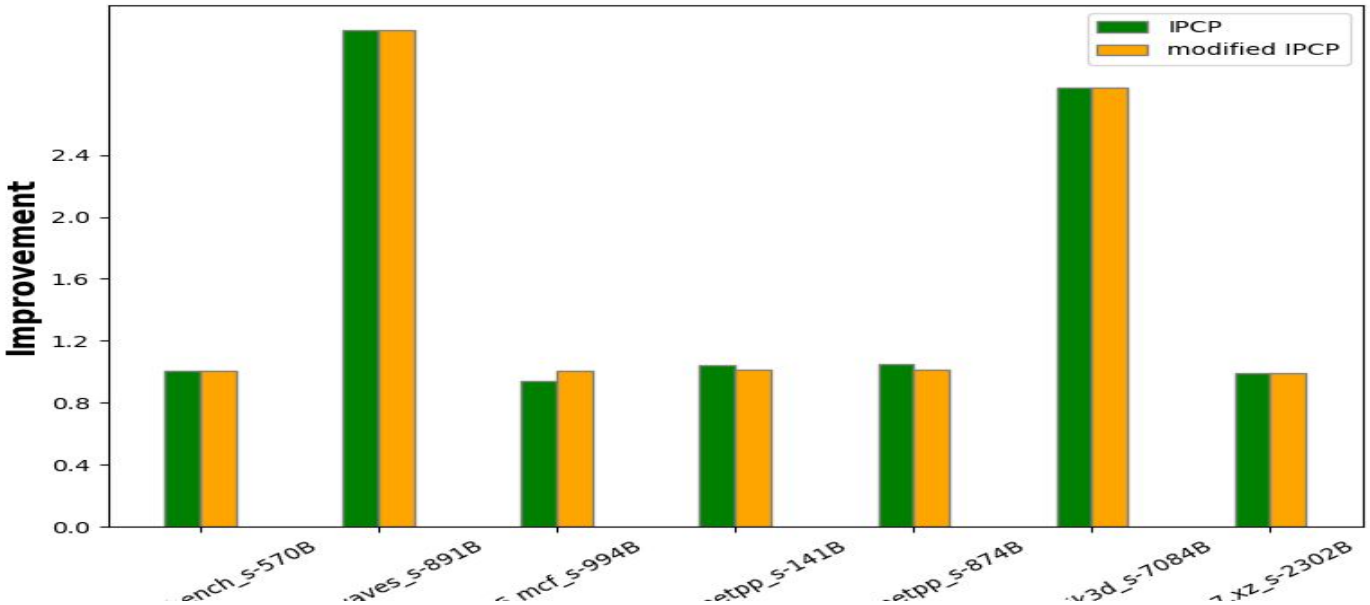
The results obtained are



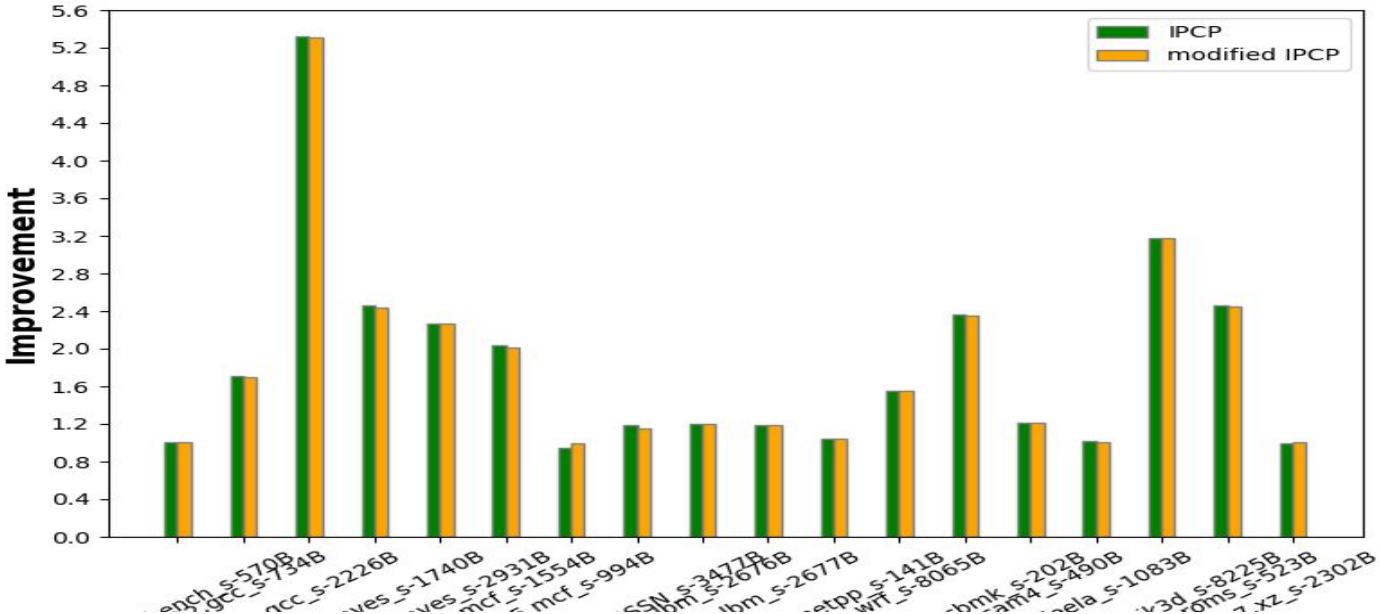
Some Observations

- IPCP gave an average improvement(in IPC) of 80.98% where as the modified code gave an average improvement(in IPC) of 78.21% over these 19 benchmarks
- The decrease in improvement after modifying the IPCP is maybe because some testbenches may find most of the prefetching data in it's own cache, hence no need to check for MSHR occupancy

Some results for NL removal



Some results for thrash protection



Conclusions

Conclusion

- We have implemented some new ideas like thrashing protection and using MSHR occupancy to decide the fill level
- We weren’t able to improve the low performing benchmarks by that much because they are irregular traces, but we got some insights and were able to get small improvements individually

Thank You