Programming Assignment 2 - Lehra Do! Prefetchers
CS 683: Advanced Computer Architecture, Autumn 2024
Computer Science and Engineering
Indian Institute of Technology, Bombay
CASPER group: https://casper-iitb.github.io/

In this programming assignment, you will be asked to implement data prefetchers, translation prefetchers, and a combination of data prefetchers in the ChampSim simulator. You must test your implementation on various benchmarks as provided and report the performance compared to the baseline with no prefetching.

PA2 Assignment invitation: https://classroom.github.com/a/2a23ZfC_
Trace files: Traces (Use IITB Account)
Download the trace files on your machine.
NOTE THAT YOU NEED GCC VERSION 7.5.0 AND AMD OR INTEL MACHINE
Please read the README carefully. If anyone asks a doubt, which is already mentioned in the README, **2 points will be deducted**.

Task 1: TLB Prefetching (5 points)

The Translation Lookaside Buffer (TLB) is a critical component, and prefetching for the TLB presents significant potential. So, the task is to implement and analyse an Arbitrary Stride Prefetcher (ASP) for the Shared Translation Lookaside Buffer (STLB).

Description:
Arbitrary Stride Prefetching (ASP) leverages the program counter (PC) to index a table known as the Reference Prediction Table (RPT). Each row in the RPT contains:
Address: The last address referenced by the instruction at the given PC.

Stride: The stride value is calculated based on the difference between consecutive addresses referenced by the instruction at the given PC.
State: A state to keep track of stride changes.

Mechanism:
Address Update: Each time the PC points to an instruction, update the address field in the RPT.
Stride Calculation: Calculate the stride based on the difference between the current and previous addresses.
State Tracking: Update the state to monitor stride changes.
Prefetch Condition: Initiate a prefetch only when there is no change in the stride for more than two consecutive references by the instruction.

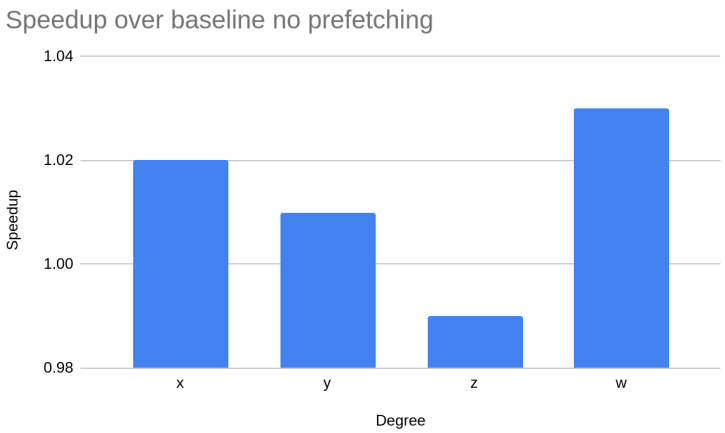
Reference: Section 2.2 https://dl.acm.org/doi/pdf/10.1145/545214.545237
https://people.engr.tamu.edu/djimenez/pdfs/isca21_dist.pdf

Note: The code for the Next-Line Prefetcher is provided in `prefetcher/next_line.l1d_pref`. Use it as a reference, but pay attention—it's designed for the cache, and you will need to adapt it for the TLB.

Steps to follow:

- In this section, you must implement ASP in ***prefetcher/asp.stlb_pref*** file
- The prefetcher should be placed beside the **STLB**
- Follow the instructions in the Champsim Repository and execute the binary with **trace1**.
- Execute for: warmup_instructions=**25000000**, simulation_instructions=**25000000**
- Report the speedup values in the tabular form and plot the overall **speedup** with different variations of prefetch degree.

  Note: Speedup of Prefetcher = IPC of Prefetcher / IPC of Baseline (without prefetching)



- Also, plot **STLB MPKI, compare it with no TLB prefetching,** and report all the values, as above.

Task 2: Data Prefetcher (5 points)

Description:

- IP Stride Detection:
  A stride represents the difference between two memory addresses being accessed. The IP stride prefetcher detects patterns in the memory addresses accessed by a specific instruction. For a given IP, if a memory address is missed, an address that is stride by a distance from the missed address is likely to be missed in the near future.
  The IP-stride prefetcher works on the same principle as the Arbitrary stride prefetcher that you implemented in the first task.

- The IP stride prefetcher fails when the stride pattern is not constant. For example, in a sequence like 1, 2, 1, 2, 1, 2, an IP stride prefetcher cannot confidently prefetch any stride because the alternating strides compete for the same entry in the IP table, resulting in zero confidence.
  To address such complex patterns, a complex stride prefetcher is required to analyse observed patterns or signatures. For instance, if the prefetcher detects a sequence like 1, 2, 1, it predicts that the following stride will likely be 2. Similarly, upon observing strides like 2, 1, 2, it predicts that the following stride will be 1. By capturing and utilising these patterns, the prefetcher can effectively reduce cache misses compared to IP stride prefetchers.

  The table below shows how the idea can be implemented (you can use other structures, too)

IP	Delta Signature	Stride
IP_1	{1,2,1}	2
	{2,1,2}	1
IP_2	{6,7,2}	5

Keep in mind not to cross the page boundary while prefetching a line.

Reference: Complex stride

Steps to follow:

- First, you need to implement the IP-Stride prefetcher in ChampSim at ***prefetcher/ip_stride.l1d_pref.***
- Follow the instructions in the Champsim Repo mentioned and execute the binary with **trace2**.
- Execute with warmup_instructions=**25000000**, simulation_instructions=**25000000.**
- Implement the complex stride prefetcher in Champsim at ***prefetcher/ complex_stride.l1d_pref.*** Run it using the same configuration mentioned above.

- **Note** that the prefetch degree cannot be changed and must be the same for all prefetching techniques.
- Plot the **Speedup** and **L1D MPKI (Total & Load)** observed over the baseline (no prefetcher) and report their numbers in tabular form. For speedup, you must compare:
    1. IP-Stride over no prefetcher
    2. Complex Stride over no prefetcher
- For L1D MPKI, compare:
    1. Baseline with no prefetcher
    2. IP-Stride prefetcher
    3. Complex Stride prefetcher

Task 3: Guldasta-e-Prefetcher (5 points)

One may think that the complex-stride prefetcher works well for all sorts of applications. You can try to evaluate the performance of **trace1** with the complex-stride prefetcher and observe the speedup. What happened here? So, the complex-stride prefetcher is not suitable for all sorts of workloads. How do we fix this? Would a different type of L1D prefetcher help us with this workload?

Description:

Your task is to implement a combination of prefetchers, specifically the IP-stride prefetcher, complex stride prefetcher, and the next-line prefetcher (prefetches the next consecutive lines). In order to decide the most suitable prefetcher for your workload, you can use each of these prefetchers for a fixed PHASE_LENGTH (number of prefetch requests) and measure the **accuracy** of each of these prefetchers. Once you are done with this learning phase, you can continue using the most accurate prefetcher for the remaining duration. This technique ensures you can always choose the optimal prefetcher for all your workloads.

Note: Apply this mechanism once the warmup is completed. You may choose any appropriate PHASE_LENGTH.

Steps to follow:

- In this section, you must implement the above technique by making changes in the ***prefetcher/optimized.l1d_pref***
- You can also make changes in the ***cache.cc*** if required.
- **Note** that the prefetch degree cannot be changed and must be the same for all prefetching techniques.
- Follow the instructions in the Champsim Repository and execute the binary with **trace1, trace2, and trace3**.
- Execute for: warmup_instructions=**25000000**, simulation_instructions=**25000000**
- Plot metrics (for all traces): **Speedup, L1D MPKI (Total & Load)**. Report their values in a tabular form as well.
- Plot and report the **Speedup** observed over the baseline (no prefetcher) to compare the below three:
    1. IP-Stride over no prefetcher
    2. Complex Stride over no prefetcher
    3. Next-line over no prefetcher
    4. Optimized over no prefetcher
- For L1D MPKI, compare for:
    1. Baseline no prefetcher
    2. Next-line prefetcher
    3. IP-Stride prefetcher
    4. Complex Stride prefetcher
    5. Optimized prefetcher

Bonus task: Task 1 + Task 2 + Task 3 (5 points)

If we activate both the data and the TLB prefetcher, the traffic will increase, leading to performance degradation. Your task is to improve overall performance with both Data and TLB prefetcher active. Feel free to use any approach to achieve a significant performance improvement. For example, you can add throttling mechanisms, etc.

Steps to follow:

- In this section, you can modify any files used in the above tasks, such as ***optimized.l1d_pref, asp.stlb_pref*** (just rename it as ***[FileName]_task4.[Type]_pref***) or other files required for your logic.
- **Note** that the prefetch degree cannot be changed and must be the same for all prefetching techniques.
- Follow the instructions in the Champsim Repository mentioned above, and execute the binary with all the traces provided.
- Execute for: warmup_instructions=**25000000**, simulation_instructions=**25000000**
- Report the numbers and plot the overall **Speedup** across all traces (baseline without prefetching) and compare it with the best speedups of task 1,2,3.

Deliverables

- Implementations in the corresponding files mentioned for the respective tasks.
- Log files with the Champsim outputs in folder ***logs/*** with nomenclature ***<Trace>_<Prefetcher>_<optional:suffix>***, <optional suffix> for any changes in parameters like prefetch degree, etc. For Example:
    - Task 1: trace1_asp_degree1.txt or trace1_asp_degree2.txt,...
    - Task 2: trace2_ip_stride.txt or trace2_ip_stride_degree3.txt or trace2_complex_stride.txt
    - Task 3: trace1_optimized.txt
    - Task 4: trace1_task4.txt
- ***Summary.pdf*** file summarizing all the tasks and the respective todos. Describe what you did and why you did it. It must include all the plots and tables of the corresponding task.

Assessment will be done via viva, and log and code files will be inspected after submission.

Submission format

- You have to push all the required deliverables to the GitHub repository with proper commit history.
- You should also submit a single tar.gz file containing your repo (please don't include trace files) with the name ***<ldap_id>_pa2.tar.gz*** on Moodle.
- The file should contain the PA2 repository code base, all tasks' implementations, generated logs and the Summary.pdf.

Late submission policy

- -2 points penalty per day (applies to late submissions on both GitHub & Moodle)