Assignment 5

Lab Overview

In the previous lab, you learned about a trace-based simulator- ChampSim. You understood how changing different parameters affect the performance of a processor. In this lab, you will be hacking the code of ChampSim to implement two features of a modern processor- LLC Bypassing and Prefetch Filtering.

LLC Bypassing is a technique where based on certain events, the processor decides whether to use LLC for the currently executing program or bypass it completely (that is, not use LLC for reading and writing data at all). There are various types of LLC Bypassing methods, but in this lab, you will be implementing an Instruction Pointer/Program Counter-based mechanism to bypass LLC.

Please note that you have to make changes to the same files you received for the last lab. However, you have to use this trace for this assignment:
https://drive.google.com/file/d/1F9vv11vVH5IbHNmSiuBrEhGa9r9sRtnU/view?usp=sharing
Submission format and what to submit will follow in Part 2.

Note: This assignment will be very crucial to help you implement your Course Project so please understand the code of ChampSim as well as you can during this assignment and ask whatever questions you might have as there won't be any help from TA's side for the Course Project.

Part 0: Understanding the problem statement (20 Points)

(10 Points)
Read the paper MadCache: https://pharm.ece.wisc.edu/papers/MadCache.pdf
Write a summary of the paper on one page.

This activity is just to make you understand how Instruction Pointer (IP) based LLC Bypassing works. MadCache is a short and simple paper. Feel free to skip Section II.D- *Multithreaded MadCache*. Interested readers can go through that part as well. For more information of set-dueling, refer to slide 12 of https://www.cse.iitk.ac.in/users/biswap/CS698Y/lectures/L8.pdf

Since the motive of this part is to make you understand the problem statement, we will be taking the doubts regarding MadCache on Piazza and in live lectures/doubt sessions. So this part is basically, understand through reading and asking doubt and summarize your understanding.

(10 Points)
Go through the code of cache.cc- handle_read( ) and handle_fill( ) in particular and answer the following questions:

- What happens in the handle_read( ) function in case of read hit? **(2 points)**
- What happens in the handle_read( ) function in case of read miss? **(2 points)**
- Why does handle_fill( ) function read requests from cache MSHR? (**2 points)**
- What are upper_level and lower_level caches? Where are they assigned in ChampSim code? **(2 points)**
- What does return_data( ) function do? **(2 points)**

This activity is to make you understand how a request for data from a particular memory location moves through the cache hierarchy. Interested readers may read and understand the complete code of cache.cc. You can either use a debugger or cout statements to understand the working of cache.cc. Please refer to the following video to understand both the ways:
gdb with ChampSim . This gdb command will come in handy:
r -warmup_instructions x -simulation_instructions y -traces path/to/trace.
You will understand what is x and y after watching the video.

For this part too, we will be taking doubts on Piazza and in live lectures/doubt sessions. While answering the questions, we will mention if your query is relevant to the assignment or not so that you can get an idea if you are going in the right direction and which part of the code you can conveniently ignore.

Part 1: Implement LLC Bypassing (25 points)

This part is your territory. If you have done Part 0 properly, you will be having an idea now what to do and where to make changes in code to do it. You have read MadCache just to understand how LLC Bypassing works, however, it is not necessary to implement the exact same design. You can make your own designs for this part but you will have to explain your design in the report that you will submit as part of this assignment. The only catch is, your design should be IP-based. Feel free to read other available material/papers online and implement.

Please make sure to comment your code properly so that anyone reading your code can understand what's going on. Also, use assert statements wherever possible to validate your assumptions. It's a good practice to catch unexpected bugs.

Marks distribution:
- Implementation **(20 points):**
    - LLC Read Bypass **(10 points)**
    - LLC Write Bypass **(10 points)**
- Comments **(5 points)**

You can use the following function signature (not necessary to use though):
```
bool llc_bypass(uint64_t ip, uint64_t cache_line)  //if true, bypass else use L3
{
        //check if the cache line is unique for the current IP. If yes, increment entries and
        counters accordingly.
        //check if for current IP, condition for the bypass is satisfied. If yes, return true else return
        false.
}
```

Part 2: Collect results and form a report (15 points)

For building ChampSim, use parameters:

Parameter	**Value**
Branch Predictor	bimodal
L1I Prefetcher	no
L1D Prefetcher	no
L2C Prefetcher	no
LLC Prefetcher	no
Replacement Policy	lru
# of CPU cores	1

For running,

Parameter	**Value**
# of warm-up instructions	10
# of simulation instructions	10

Since you are making a change in processor design, you will have to specify if there's a performance improvement or degradation with respect to the older design(where there was no LLC bypassing). The older design is called the **baseline**. You will have to submit the following things:

- Collect IPC of baseline and keep the result files (files that are generated when you run champsim on a trace) in a folder called *baseline_no_bypass_result*. **(3 points)**
- Implement LLC Bypassing and collect IPC with LLC Bypassing. Keep the result files in a folder called *llc_bypass_result.* **(3 points)**
- Submit a report which will have the following content **(9 points)**:
    - A detailed description of the design you are using for LLC Bypass. Please use diagrams and flowcharts if needed. **(5 points)**
    - List of all the source files you have edited or added to implement LLC Bypass. **(2 points)**

    ○ Result of what was the IPC improvement or degradation with respect to the baseline. **(2 points)**

Part 3: Prefetching (25 points)

In this part of the assignment, you will be implementing prefetch filtering/throttling for simple IP-stride prefetcher. Before jumping into implementation, you can have a look at the handle_prefetch() function in cache.cc.

You already know the basics of prefetchers by now. You also know how a stride prefetcher works. **IP-stride** prefetcher is an extended version of the simple stride prefetcher which handles stride patterns based on instruction pointers. It maintains a table of previous addresses accessed by a list of instruction pointers. When the same instruction is executed again, a stride is calculated for the address accessed and a prefetch request is made based on it.

What you will be implementing is a throttling mechanism based on prefetch accuracy for the IP-stride prefetcher(which will be provided). Again this will be on a per IP basis, where you will need to change the prefetch degree dynamically based on some threshold based on prefetch accuracy. You will need to modify the IP_TRACKER(prefetcher table) accordingly.

Marks distribution:

- Implementation **(20 points)**
- Comments **(5 points)**

You can use the following function signature (not necessary to use though):

```
void prefetch_throttle(uint64_t ip)
{
        //update the prefetch_degree based on the prefetch accuracy for this ip
}
```

Part 4: Collect results and form a report (15 points)

For building ChampSim, use parameters:

Parameter	Value
Branch Predictor	bimodal
L1I Prefetcher	no
L1D Prefetcher	ip-stride
L2C Prefetcher	ip-stride

LLC Prefetcher	no
Replacement Policy	lru
# of CPU cores	1

For running,

Parameter	**Value**
# of warm-up instructions	10
# of simulation instructions	10

Find the performance improvement with prefetch filtering for different thresholds with respect to older design(with no prefetch throttling). You will have to submit the following things:

- Collect IPC of baseline and keep the result files (files that are generated when you run champsim on a trace) in a folder called *baseline_no_throttling_result*. **(3 points)**
- Implement prefetch throttling and collect IPC with prefetch throttling. Keep the result files in a folder called *prefetch_throttling_result.* **(3 points)**
- Submit a report which will have the following content **(9 points)**:
    - Normalized IPC improvement wrt thresholds used. **(4 points)**
    - Prefetcher coverage vs thresholds used. **(3 points)**
    - List of all the source files you have edited or added to implement Prefetch Throttling. **(2 points)**

Note: Group leader must make sure that submitted report has a proper division of labor mentioned. It should properly state which team member did what? Biswa will ask questions from team members based on this division. If you have mentioned the division honestly, there's nothing to worry about.

Please use following directory structure for submission:

```
<your_group_number>/
    |-- Champsim/                          # All the code files
    |-- report.pdf
    |-- baseline_no_bypass_result/         # .txt files of results
    |-- llc_bypass_result/                 # .txt files of results
    |-- baseline_no_throttling_result/     # .txt files of results
    |-- prefetch_throttling_result/        # .txt files of results
```

(Directory structure of your submission)

Compress the directory and name it the same as your group name (for example, `<group_name>.tar.gz`) and submit it on Moodle.