CS 6354: Graduate Computer Architecture Fall 2024

Homework 5

HW Out: Nov 8 **HW Due:** Nov 26th *(100 points)*

Student Name: _____

Collaborator Name(s): _____

Instructions

Via this homework, you will be testing your code that you wrote (as part of HW4) in *cache_sa.py* on real traces (applications). There is a new file called *memory.py*, which has the class for Memory (DRAM). Please understand the existing code before making changes (We have made some changes from the previous version).

1. Since we have made some minor changes to the *cache_sa.py* file COPY your function definitions from HW4 into the current file before you run the experiments.
2. To simulate a cache, go the directory containing the *sim_sa.py* file and use the command ***python3 sim_sa.py*** <***PATH TO TRACE***> <***SIZE OF CACHE***> <***WAYS***> <***BLOCK SIZE***> <**TRACE_ELEMENTS**>.

We will be using traces from real applications in this assignment. The real traces (applications) can be downloaded from here: `https://github.com/adwaitjog/ramulator`

More information on real traces can be found in *README.md*. In this homework, we will be mostly using the following 4 traces, but you are welcome to use more real traces.

1. 462.libquantum.gz
2. 464.h264ref.gz
3. 403.gcc.gz
4. 445.gobmk.gz

Deliverables

1. **2 python files**, **cache_sa.py** for the Cache and **memory.py** for task 1. DO NOT CHANGE THE NAME OF THE PYTHON FILES.
2. **1 report** containing,
   (a) Output of the Code when you run the real world traces.
   (b) Document your observations (graphs are preferred) and submit the PDF. The recommended length of this report is three pages.

Showing your work is not optional and your final score will be highly dependent on the quality of the report.

1 Tasks

This homework has the following three tasks:

1.1 Completing simple cache-memory simulator: 20 pts

Complete the determine_miss_penalty function in *memory.py*. The function is a maximum of five lines of python code and you are welcome to use other methods defined already in the Memory class definition (understand them first before using!).

1.2 Verifying with real traces: 40 pts

Verify the output traces of your complete memory-cache simulator with the ones given in the resource folders (resources/output_files). If they do not match, debug your code (in cache_sa.py or memory.py).

1.3 Cache design space exploration: 40 pts

Find how the following three parameters affect cache miss rates.

1. cache size
2. block size
3. associativity

Our goal is to come up with a "generic" cache design that works well on average for many applications (traces). After you have finished Tasks 1 and 2, come up with a combination of associativity and cache block size that works the best on average for as many real traces as possible. The metric that we will be using for measuring goodness is *CPI_stall* (lower the better). The *CPI_stall* equation is already in *sim_sa.py*. The *CPI_stall* for a cache design X (i.e., combination of cache size, associativity, and block size), is the geometric mean of *CPI_stall* numbers that you get with different traces using the same cache design X.

To limit the design space, you can assume the following constraints. You should test on at least the four traces

1. 462.libquantum.gz
2. 464.h264ref.gz
3. 403.gcc.gz
4. 445.gobmk.gz

Keep the cache size constant across all the experiments: 16384 (16KB), which is a reasonable size of L1 caches for modern processors. In terms of associativity, you can explore 1-way (direct-mapped), 4-way, 8-way, 16-way, and 32-way. In terms of block size, you can explore 128B, 256B, 512B, 2048B. Additionally, you can also consider like: 4096B, 16384B. Some of the cache block sizes here (especially, > 512B) are unrealistic but provides interesting insights. Analyze your results with data/graphs.

1.4 Example command line with real trace

```
python3 sim\_sa.py 403.gcc.gz 16384 16 128 1000
```

- where 403.gcc.gz is a trace file stored in *../traces*
- 16384 is the cache size in bytes
- 2 is the number of ways
- 128 is the number of bytes per block (i.e., BlockBits = 7: 2 for byte offset, 5 for block offset as there are 32 words)
- **NEW!** 1000 is the number of simulated lines from the trace file. Each trace file has 100s of thousands of lines.