Project: Prefetchers

1 General Instructions

- ◇ start by accepting the assignment link, which will create the repo for you on your github. Then clone this repo into the VM.
- ◇ Make sure you use the provided VM for developing and running the labs. We will not help debugging any other setup.
- ◇ In the lab room machines, you can login in to the VM with your group number. You will be asked to enter a password for the first time. Feel free to copy the VM into your own laptop to work on the labs at your time.
- ◇ clone the repository into a folder named macsim-4dm4-lab4; all instructions in this document assumes this folder as the main repo folder.
- ◇ In addition to the code modifications and output files, you are asked to prepare a pdf report of all the steps and answering the questions within the lab document. The first page of the report should be a declaration of contribution where each of the two members of the group lists clearly what they have done.
- ◇ All the files you are asked to produce or modify (including source, configuration, trace files, output files, and report) should be placed under labs/lab4 folder.
- ◇ The submission of the code, output files, and the lab report should be done by pushing to the repository before the deadline time.
- ◇ In addition to the declaration of contribution, we will be using github commit history to track the contributions of the group. Therefore, it is very critical to commit and push updates one at a time and do not wait until you finish the full lab to push once. This will be considered a red flag.
- ◇ Finally, please make sure you only push the required files, the report, the parameter files you modify for each step, plus any source files you modified. Log files and benchmark source files are not required.

2 Submission Deadline

The deadline for lab4 is **Tuesday Dec 5th, 11:59PM**. Please note that this is a programmed deadline in the environment, so make sure you submit in time since you will not be able to submit after that dictated deadline.

2.1 Importing the starter code of the lab

You should follow exactly the same process you have been following in past labs to import the starter code from the following invitation link and to create the project:

https://classroom.github.com/a/blYAWIW6

3 Introduction

In this project you will be working to implement a hardware prefectcher. You will be given a basic cache simulator with an interface to a prefetcher. Your task will be to implement the prefetcher interface with a prefetching algorithm of your own choice. The effectiveness of your prefetcher will be tested against a baseline prefetcher. You will also compete against your fellow classmates in other groups for certificates and and prizes!

4 Cache Configuration

Your prefetcher will work in the context of a well-defined memory hierarchy. The memory system is already implemented (in C++) and a part of the code in this repo.

- ◇ The Data Cache has the following stats:
  - 32KB Size
  - 2-way Set Associative
  - 16 byte line size
  - write-through
  - no-write-allocate
  - 1 cycle hit time
  - 10 entry write-buffer
- ◇ A L2 cache (data only) has the following stats:
  - 256KB Size
  - 8-way set associative
  - 32 byte line size
  - write-back
  - write-allocate
  - 20 cycle access time (fully pipelined)

Because it is fully pipelined, it is possible to issue 1 request (either from the data cache or the prefetcher) to the L2 cache per cycle. Main memory has an access time of 50 cycles and is not pipelined (i.e. only one request can be handled at a time). Accesses to main memory are held in a FIFO queue with 10 entries. If the FIFO fills up, the L2 will no longer accept requests.

5 Prefetcher Interface

Your main task in this project will be to implement a prefetcher using the given prefetching interface. The system controller provides information about all loads and stores that are issued by the CPU. The information that is provided includes the effective memory address, PC of the memory instruction, and whether the instruction was a load or a store. You may use this information in any way that you see fit. During all cycles where the CPU is not issuing a request to the L2 cache, the system controller will query the prefetcher for any memory requests that it may have. While your prefetcher may have many requests queued internally, the system will service a maximum of 1 per cycle. After the prefetching request has been satisfied (either from the L2 or main memory), it will be placed in the Data Cache.

The file prefetcher.h pre-defines four functions that must be implemented:

- ◇ bool hasRequest(u_int32_t cycle) - Value returned indicates whether there is a request waiting from the prefetcher. This function is used as a gate before getRequest() is called.
- ◇ Request getRequest(u_int32_t cycle) - Returns a request to be sent to the L2 cache. Only the addr field of the Request struct will be used.
- ◇ void CompleteRequest(u_int32_t cycle) - This function is called as soon as the last request is successfully sent to the L2 cache.
- ◇ void cpuRequest(Request req) - This function is called after a CPU request is handled. The request gives the addr, pc, cycle it was issued, and whether it was a hit or miss in the D cache.

The prefetcher is called only from the main.C file. While you are free to examine all parts of the provided memory system, the only modifications you should make is to the prefetching interface contained in the files prefetcher.h and prefetcher.C.

To aid in your understanding of the prefetcher interface, we have provided a sample prefetcher implementation. This simple prefetcher waits for misses on the D-cache and then tries to prefetch the next block in memory.

You can find the sample prefetcher in the sample-pf folder.

6 Prefetcher Constraint

In addition to the constraint that only a single request can be serviced per cycle, you will have one further constraint: the amount of state saved in the prefetcher. The amount of state saved in the prefetcher may not exceed 4KB. Your source code must clearly indicate which variables are used as state. Furthermore, you will need to provide a detailed accounting in your project report of how much state is kept.

7 Trace Format

The memory hierarchy will be simulated using trace files generated by the Pin binary instrumentation tool. Each line in the trace file refers to a memory access and includes the following four pieces on information:

- ◇ Whether the instruction is a load or store
- ◇ PC of memory instruction (32 bits)
- ◇ Effective Memory Address (32 bits)
- ◇ Number of non-memory ops (i.e. computation instructions) since the last memory operation

For your report, you should test your prefetcher on traces available under traces folder. However, TAs will test your prefetcher with another set of memory traces for the prefetching competition. Please consider making your design working for general cases. In addition, if TA cannot reproduce the experimental result in your report, your project will be considered as fail.

You may wish to extend the simulator to collect more stats. This is not required, and your prefetcher should not depend on these modifications.

8 Simualtion Statistics

The memory system provided will output several statistics about the performance of the system. They will help you understand how your prefetcher is performing and why. The stats include:

- ◇ Total Run Time
- ◇ D-Cache Total Hit Rate
- ◇ L2 Total Hit Rate
- ◇ Average Memory Access Time
- ◇ Average Memory Queue Length
- ◇ D-Cache-to-L2 bandwidth utilization
- ◇ Memory bandwidth utilization

These stats will be placed in the file mem.trace.out, where mem.trace was the input file used.

9 Prefetcher Starting Points

Here are some papers to get you thinking about different approaches to prefetching. You are not required to choose an algorithm from these papers, but they can provide some useful starting thoughts. Their bibliographies will provide pointer to other papers on the topic.

- ◇ A Primer on Hardware Prefetching: An Introductory book to HW prefetching, you can access it through McMaster here: https://link-springer-com.libaccess.lib.mcmaster.ca/book/10.1007/978-3-031-01743-8

- ◇ Spectral prefetcher: An effective mechanism for L2 cache prefetching http://prod.tinker.cc.gatech.edu/journal/spectral.pdf
- ◇ A stateless, content-directed data prefetching mechanism https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.442.3212&rep=rep1&type=pdf
- ◇ Prefetching using Markov predictors https://ieeexplore.ieee.org/document/752653/
- ◇ Memory Prefetching Using Adaptive Stream Detection https://www.cs.utexas.edu/~lin/papers/micro06.pdf
- ◇ Reducing memory latency via non-blocking and prefetching caches https://dl.acm.org/doi/10.1145/143371.143486

10 Evaluation

Your grade for the project will be based on your write up the prefetcher you implemented. The performance of your prefetcher is less important than your discussion of how the prefetcher works and why it performs the way it does.

- ◇ 30% Report
- ◇ 30% If your code can compile and is able to work correctlt
- ◇ 60% Performance: we will use the given "sample prefetcher" as the baseline and measure the speedup (S) of your prefetcher. Your grade in the part is then evaluated as min((S-1),1). For example, if you get a speedup of 2, you will get full credits on this part.

11 Deliverables

1. Source Code: You are asked to only modify two source code files: prefetcher.h and prefetcher.C.
2. Output Stat files: For all the given traces, you need to provide the output stat files for each one, put them under `out` folder.
3. Report: Your report for project will consist of the following sections:
   - ◇ Declaration of Contribution Statement per group memeber.
   - ◇ Description of Prefetcher: One or two paragraphs describing the prefetching algorithm that you are using. If your prefetcher is based on an existing design, you must explicitly state the original source. While you are free to implement existing designs, failure to provide a citation will result in you receiving no credit for the project. Your description should include the rationale for the design, supported, where appropriate data evaluating "teaks" or changes you made to improve performance.
   - ◇ State Accounting: How much state your prefetcher uses as well as a detailed description of how and where the state is used. For example, if you have a prediction table, you should list its size as well as its layout.

- ◇ AMAT Graph: A bar chart showing the average memory access time (AMAT) of your prefetcher on the provided traces. Your report should be in PDF format and also placed under the `out` folder.