CSE 520 Computer Architecture – Spring 2019
Programming assignment 1 (100 points)

Task 1: Memory Hierarchy Performance Measurement

Write a C program running on Linux to access (read, write) memory in different pattern (linear, random), and measure the average access time of L1, L2, L3 cache and main memory. You can measure the average access time to an array. If the array can be fit into L1 cache, then the access time of the array denote that of L1 cache. By varying the array size you can get the data similar as figure 1. Describe your method and report your data in tables and diagrams. Then answer the following question.

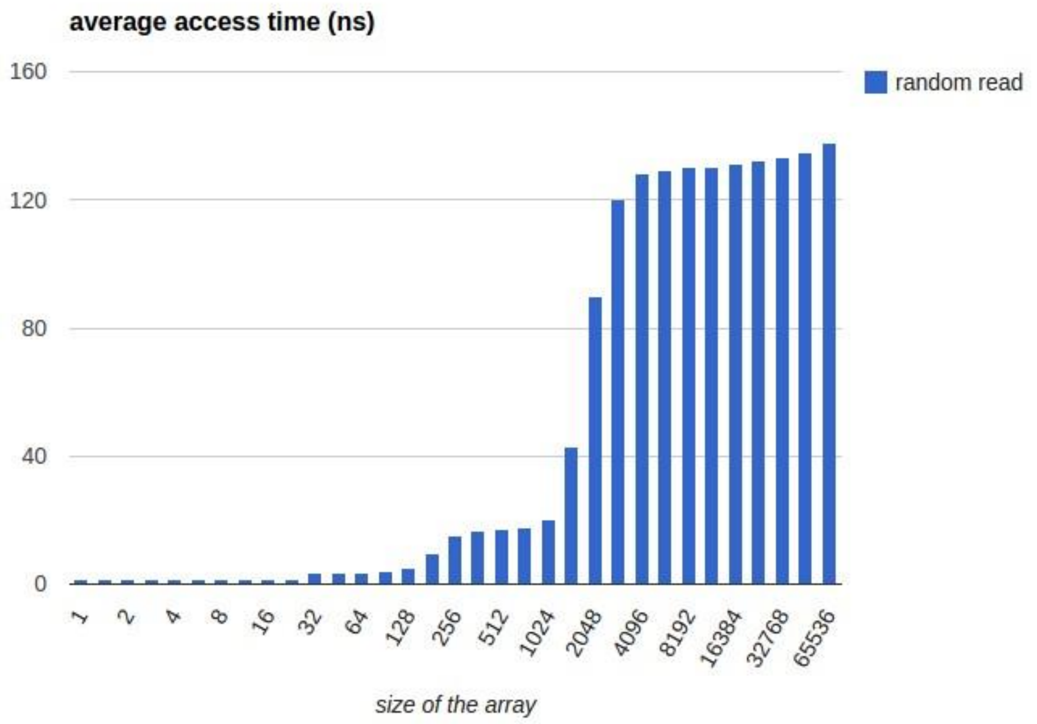

Figure 1: average access time of a single element in arrays of various size (random read)

(a) Is this average access time represents the *latency* or *throughput* of memory hierarchy? Why? How did you mitigate the overhead in measurement, or how did you improve the accuracy of your measurement?
(b) What makes the access time different for linear access pattern and random access pattern? Is there difference for read and write? Why?

Hint 1: The performance of the memory hierarch under the impact of the (spatial and temporal) locality of accesses is characterized as “Memory Mountain”. Please refer to section 6.6.1 of the book "Computer Systems: A Programmer's Perspective" by Randal E. Bryant and David R. O'Hallaron. A pdf file of the book can be found at https://piazza.com/class_profile/get_resource/j7ly9riuca97on/ja86xbbpp0b73b.

Hint 2: If you have no idea of how to do the measurement, there are a few options you might want to try. You could simply insert clock() function in your code to measure the time, or use “time” command in the Linux to measure the whole program execution time. However, do not try to use these methods to measure a program running less than 1 millisecond.

Another method is to use “perf” tool. This tool gives you more information such as number of instruction and total count of cycles. It has also been considered much more accurate. However, this tool is restricted to what kind of performance counters the CPU has been implemented. For example, with Intel i7-4770, perf is able to report the total number of memory write instruction, while in AMD A8-7410 this data is unavailable.

Hint 3: You may want to create an array of structure where the size of the structure is the same as the LI cache line size. Thus, consecutive accesses to array elements will go to different cache lines. Also, for random memory access pattern, you can use the “pointer chasing” approach to ensure that the successive access can only be done after the current one.

Hint 4: You need to repeat the operation many times and measure the accumulated execution time, because in general the computer and operating system are unable to measure time in very high precision (microsecond level is OK). To automate the measurement process, you can construct and run a bash script that receives command line arguments or gcc defining macro (-D) to pass different array sizes for measurement.

Task 2: Reproduce Task 1 Using Gem5 Simulator

Run the program you wrote in the task 1 on Gem5 simulator. You need to configure your simulator as following.

CPU type:	DerivO3CPU
L1 instruction cache:	2-way set-associate, total size 32KB, LRU
L1 data cache:	4-way set-associate, total size 64KB, LRU
L2 unified cache:	16-way set-associate, total size 256KB, LRU

Other system settings should remain unchanged (default setting), including cacheline_size=64 bytes.

Assume the CPU in the simulator running in 1000MHz, you could estimate the program execution time using the simulation cycles. Please have the diagram and table similar to what you have in Task 1. The table should also contain the cache hit rate among different cache levels.

Do not perform any measurement method mentioned in the Task 1 for simulation run. Just use the statistics data from the simulator.

Due Date

This assignment will be due at 11:59pm on Jan. 29.

What to Turn in for Grading

- Develop a report to show the memory mountain(s) diagrams for read/write and linear/random accesses in Tasks 1 and 2. Also, your answers to questions (a)-(b) should be included in the report.

The report should be limited to 3 pages for each task. Don’t forget to add your name and ASU id in the report.

- Create a working directory to include your source files, makefiles, readme, and the report (in pdf). Your source files should include all test programs for the assignment. Comment your source files properly and provide precise description on how to compile your source code in reame file. (please clean up all object code from your submission and we will recompile your code using your makefiles.)
- Compress the directory into a zip archive file named **cse520-lastname-firstinitial_assgn01.zip.** Note that any object code or temporary build files should not be included in the submission.
- Submit the zip archive to Canvas by the due date and time.
- Failure to follow these instructions may cause an annoyed and cranky TA or instructor to deduct points while grading your assignment.
- There will be 20 points penalty per day if the submission is late. Note that submissions are time stamped by Canvas. If you have multiple submissions, only the newest one will be graded. If needed, you can send an email to the instructor and TA to drop a submission.
- The assignment must be done individually. No collaboration is allowed, except the open discussion in the forum on Canvas. The instructor reserves the right to ask any student to explain the work and adjust the grade accordingly.
- Here are few general rule for deductions:
    - No make file, or compilation error -- 0 point for the assignment.
    - Must have “–Wall” flag for compilation – 5 points deduction for each warning.
    - 20-point deduction if no compilation or execution instruction in README file.
    - Source programs are not commented properly – 10-20 points deduction.
    - Inclusion of any object files in your submission – 20 points deduction

- ASU Academic Integrity Policy (http://provost.asu.edu/academicintegrity), and FSE Honor Code (http://engineering.asu.edu/integrity) are strictly enforced and followed.