Homework 2

PROFILING THE IMPACT OF CACHING, MEMORY ACCESSES, AND CHOICE OF DATA STRUCTURES

VERSION 1.0

The objective of this assignment is to demonstrate the effects of caching, pre-fetching, the memory hierarchy, and how different data structures affect accessing memory. The programming assignment should be implemented in Java. You are required to work alone on this assignment. The assignment accounts for 7.5% towards your cumulative course grade.

This assignment may be modified to clarify any questions (and the version number incremented), but the crux of the assignment and the distribution of points will not change. If there are any changes to the assignment, all changes will be documented in the "Change History" section of this assignment.

1 Assignment Description

The goal of this assignment is for you to gain a better understanding of the effects of the memory hierarchy. As part of this assignment, you will be responsible for measuring the cumulative effects of registers, caching, and fetching from main memory. The programming assignment should be developed in Java version 8, and your classes must reside in the package "cs250.hw2".

Your program will be provided with 3 arguments at the command line. You are required to perform a set of experiments that are configured using the specified arguments. The points distribution for each task and the restrictions (and accompanying deductions) are specified in the grading section of this assignment.

We will compile and run your program using the commands below, replacing "<size>", "<experiments>", and "<seed>" with numbers as described in the tasks. See the "Example Outputs" section for more examples on how your program will be run. Your code must compile and run on the Linux machines in the CSB 120 computer lab, using Java version 8 (the default on these computers). It is highly recommended to develop your code using VS Code, either directly on the machines or through the Remote SSH extension in VS Code. Other IDE's or text editors may not be supported by the instructors.

Command line compilation: `javac cs250/hw2/Memory.java`
Command line execution format: `java cs250.hw2.Memory <size> <experiments> <seed>`
Command line execution example: `java cs250.hw2.Memory 25000000 20 42`

Note: the arguments checked must come from the "args" of the main method. Using the method "Scanner.nextLine()" (or a similar method) to read the numbers will result in a deduction.

1.1 Task 1

The first task involves contrasting the performance of programs with and without caching. In particular, you will be working with the `volatile` keyword in Java. The `volatile` keyword informs the compiler that the variable should not be cached and accesses should always go to main memory.

You will run two separate loops, one when the loop variable is `volatile` and one for the loop variable is not `volatile`. You will loop <size> number of times for both loops, so if size is 5 your loops will run 5 times.

Your loop must do something on top of incrementing the loop variable, the loops will maintain a running total of the addition and subtraction operations using the loop variable. The choice of whether you perform an addition or subtraction to the `runningTotal` is based on whether the loop variable is odd or even for the given iteration. If the loop variable is even (e.g., 10) then you should add the loop variable to the `runningTotal`; if the loop variable is odd (e.g., 37) then you should subtract the loop variable from the `runningTotal`. To cope with potential overflows/underflows `runningTotal` should be a long variable type.

You will do this <experiments> number of times, then calculate the average time (totalTime/experiments) that it took to perform operations on `runningTotal` for both loops. You will also calculate the average sum for all experiments (totalSum/experiments) for both loops.

You will then output the statistics for each of these loops. You will print out the average time taken to perform the loop with the regular and volatile variable. Then you will print out the running totals that each of these loops maintained. These numbers should be the same as both loops run for the number of times(<size>). Look at the example output below to see more on how these numbers are outputted to the terminal.

Produce a short report **(450-500 words)** with graphs and/or tables describing the observed behavior when using the `volatile` keyword versus without.

1.2 Task 2:

Allocate an array with size `<size>` and fill it with random numbers using the `<seed>` to seed the random number generator. To get the most noticeable effect use the `Integer` type rather than the primitive `int` type. For `<experiments>` times do the following:

Calculate the time to access each element in the first 10% of the array and a single random element in the last 10% of the array.

Next, calculate the sum of each of the elements accessed and report the following averages:

1. Time to access a single element in the first 10% of the array.
2. Time to access a single random element in the last 10% of the array.
3. Average sum of the elements.

It may be helpful to store the total access time with two variables, one for the first 10% access time, than with one for the single element last 10% access time. This will make it easier to find the average time for all experiments.

Note: Do not use Random.nextInt(lowerBound, upperBound) as that is exclusive to later versions of Java and will not compile on our version of Java. You will have to figure out another way to get an index within the last 10% of the array.

1.3 Task 3:

Allocate a TreeSet and LinkedList both with size `<size>` and fill both structures with the range of numbers [0, size).

For `<experiments>` times do the following:

- Calculate a random number in the range [0, size) and time how long the `.contains()` method takes to find if the element exists in the structure.

Report the average time for each of the structures to find if the element exists (totalTimes/experiments).

Produce a short report **(450-500 words)** with graphs and/or tables describing the observed behavior when using TreeSet versus a LinkedList.

2 Example Outputs

Note: Times will vary depending on the computer, if running on computers in CSB 120 lab, times should be within that decimal position. So if the example says 0.15, any times from about 0.1-0.9 should be alright, times that are greater than 1 magnitude difference have the chance of a deduction depending on how off it is.

Command: java cs250.hw2.Memory 25000000 1 42

Task 1

Regular: 0.02633 seconds

Volatile: 0.16463 seconds

Avg regular sum: -12500000.00

Avg volatile sum: -12500000.00

Task 2

Avg time to access known element: 15.13 nanoseconds

Avg time to access random element: 646.00 nanoseconds

Sum: -1005470868.00

Task 3

Avg time to find in set: 69055.00 nanoseconds

Avg time to find in list: 83721555.00 nanoseconds

Command: java cs250.hw2.Memory 25000000 20 42

Task 1

Regular: 0.04676 seconds

Volatile: 0.16412 seconds

Avg regular sum: -12500000.00

Avg volatile sum: -12500000.00

Task 2

Avg time to access known element: 15.27 nanoseconds

Avg time to access random element: 146.75 nanoseconds

Sum: -834230.20

Task 3

Avg time to find in set: 9709.15 nanoseconds

Avg time to find in list: 99872813.60 nanoseconds

Command: java cs250.hw2.Memory 25000000 200 42

Task 1

Regular: 0.04752 seconds

Volatile: 0.16394 seconds

Avg regular sum: -12500000.00

Avg volatile sum: -12500000.00

Task 2

Avg time to access known element: 15.23 nanoseconds

Avg time to access random element: 125.23 nanoseconds

Sum: -9199369.91

Task 3

Avg time to find in set: 6214.98 nanoseconds

Avg time to find in list: 75832812.82 nanoseconds

3 What to Submit

Use the CS250 *Canvas* to submit a single .tar or .zip file that contains your Java source code, plus a README.txt file. When the archive is opened, the directory structure should be as follows:

- cs250
  - hw2
    - Memory.java (and any other .java files needed for the program)
    - README.txt

The README.txt file must contain a description of each file submitted and any other information you feel that the TAs will need to grade your program.

Other files (such as the ".class" files from compiling your code, or your "hw1" folder from the previous assignment) are allowed to be present and will be ignored.

Filename Convention: The archive file should be named as <FirstName>-<LastName>-HW2.tar. E.g., if you are Cameron Doe then the tar file should be named Cameron-Doe-HW2.tar. Note: Canvas sometimes adds an extra "-1", "-2", etc. to the file name, which is OK.

Tar File Command Format: `tar -cf <FirstName>-<LastName>-HW2.tar cs250`
Tar File Command Example: `tar -cf Cameron-Doe-HW2.tar cs250`

Zip File Command Format: `zip -r <FirstName>-<LastName>-HW2.zip cs250`
Zip File Command Example: `zip -r Cameron-Doe-HW2.zip cs250`

Note: we highly recommend using one of the commands above for creating your archive for submission. Creating an archive through your operating system's GUI (or another GUI program) may not zip the file correctly, potentially resulting in a deduction on the assignment.

4 Grading

The assignments must compile and function correctly on machines in the CSB-120 Lab. Assignments that work on your laptop on your particular flavor of Linux, but not on the Lab machines are considered The assignments must compile and function correctly on machines in the CSB-120 Lab. Assignments that work on your laptop on your particular flavor of Linux, but not on the Lab machines are considered unacceptable. The following commands can be used to compile your code, execute the program, and then archive your code either as a .tar or a .zip file (either format will be accepted).

Command line compilation: `javac cs250/hw2/Memory.java`
Command line execution format: `java cs250.hw2.Memory <size> <experiments> <seed>`
Tar File Command Format: `tar -cf <FirstName>-<LastName>-HW2.tar cs250`
Zip File Command Format: `zip -r <FirstName>-<LastName>-HW2.zip cs250`

This assignment will contribute a maximum of 7.5 points towards your final grade. The grading will also be done on a 7.5 point scale. The points breakdown is as follows:

2. **2.5 points each** for correctly performing Tasks 1, 2, and 3.
   Note that the report writing component accounts for 1 point in each of the tasks, except for Task 2. **NOTE:** that the formatting for each task should match the example outputs exactly.

Deductions:

0.5 points each for not following the specified formatting for Tasks 1, 2, and 3.
2.5 points if programs need manual intervention when running.
Note this includes:
- Compilation errors
- Any extra steps required to run the program. Example outputs include the command that should be used to generate that output.

You are required to **work alone** on this assignment.

5 Late Policy

Please check the class policy on submitting late assignments. You are allowed to submit assignments up to 7 days late with a 10% deduction for each day that it is late.

6 Version Change History

This section will reflect the change history for the assignment (if needed) after the first public release of the assignment. It will list the version number, the date it was released, and the changes that were made to the preceding version. Any changes to the first public release are made to clarify the assignment; the spirit or the crux of the assignment will not change.

Version	Date	Comments
Final	9/13/2023	Final Version of Assignment.