Homework 2

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

VERSION 1.1

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.

DUE DATE: Wednesday, March 6th @ 8:00 pm

1 Description of Task

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, 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.

Command line execution: `java cs250.hw2.Memory <size> <experiments> <seed>`

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.

To profile the impact of caching, you will be contrasting the performance of loops when the loop variable is marked `volatile` and when the `volatile` keyword is not used for the loop variable.

Your loop 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.

For <*experiments*> times calculate the average time taken to perform `runningTotal` when the loop variable ranges from [0, `<size>`).

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, maintain a sum of each of the elements accessed and report the average across experiments for each of the following:

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. Sum of the elements

Produce a short report **(450-500 words)** with graphs and/or tables describing the observed behavior when accessing elements at the prescribed portions 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.

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

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 .zip file that contains:

- <FirstName>-<LastName>-Writeup.pdf
- Your java source codes, matching this directory structure:
    - cs250
        - hw2
            - Memory.java
- A README.txt file containing a description of each file and any information you feel the TAs need to grade your program.

Filename Convention: The archive file should be named as <FirstName>-<LastName>-HW2.zip . E.g., if you are Cameron Doe then the zip file should be named Cameron-Doe-HW2.zip.

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 unacceptable.

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:

2.5 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.
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 2 days with a per-day deduction of 7.5%.

6 Version Change History

This section will reflect the change history for the assignment. It will list the version number, the date it was released, and the changes that were made to the preceding version. 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
1.0	2/13/2024	First public release of the assignment.
1.1	2/26/2024	Clarify "average sum" of elements across the experiments in Task 2