|  |  |  |
| --- | --- | --- |
| **CPSC 245 OPERATING SYSTEMS** | | **FALL 2024** |
| **Java Thread Calculations – Questions [20]** | **Due: With Program** | |

Add your information to this document in **red**. Do not to change the forms or fonts, just enter your information/answers and leave them aligned as much as possible (they should expand as necessary). No microscopic fonts please :) – your goal should always be effective communication (readability in this case), here and on the job. The analysis is based on running your program on a Windows PC1.

Read this carefully, the numbers you use for your trial runs will differ from mine.

**Part I** – Functionality Disclosure (delete one response, leave the correct one)

Program tested and fully functional: **YES** (If it’s *not* 100% YES, you must delete YES, and leave NO)

**If NO describe what is not working in detail:**

**Part II** – Hardware Platform Details

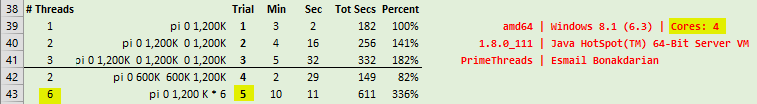
The last column is the product of the two previous columns – **you must use a machine with at least 4 total threads[[1]](#footnote-1)** (i.e., TT, **# OF CORES \* # THREADS PER CORE** >= 4). Java will report these as “cores”. All of your trials must be done on the same machine[[2]](#footnote-2) without any other programs running to avoid throwing off your results/analysis.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TOTAL RAM (GB)** | **CPU MODEL** | **# OF CORES** | **# THREADS PER CORE** | **TOTAL # OF THREADS (TT)** |
| 32 | AMD Ryzen 7 5800H with Radeon Graphics | 8 | 2 | 16 |

**Part III** – Analysis Questions

You have to pick a value for your upper limit (**N**) that takes about 3 minutes to run with a single thread, i.e., command line args are **0** and **N**, and then use it for your 5 runs, a.k.a. trials (you may have to experiment to find this value). All runs must be in the bash shell and must use the **time** command.

Here is an example of one set of 5 runs to help you. My **N** is 1,200,000[[3]](#footnote-3), and trial #5 is done with **1.5 x** **TT** (e.g., I have TT = 4 total number of threads – reported as cores by Java, so I created 6 (1.5 x TT) threads via command line arguments). My first run defines what 100% is, the other values are computed by dividing the total seconds against the total seconds of the first trial (182 seconds in my case).



Note: Trial # 4 uses command line args **0** to **N/2** and **N/2** to **N**, trial #5 repeats args **0** to **N TT x 1.5** times. I.e., the pair 0 1,200 K is repeated 6 times on the command line for me.

Use this form to record your information – I’ve filled in much of your information already.

**N** (runs for @ 3 minutes) = \_\_1834600\_\_\_ Total # of Threads (**TT**) = \_16\_\_ **TTx1.5** = \_\_24\_\_\_\_\_

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **# Threads** | **Args** | **Trial** | **Mins** | **Secs** | **Total Secs** | **Percent** |
| 1 | **0 N** | #1 | 3 | 0 | 180 | 100% |
| 2 | **0 N 0 N** | #2 | 3 | 30 | 210 | 117% |
| 3 | **0 N 0 N 0 N** | #3 | 3 | 50 | 230 | 128% |
| **2** | **0 N/2 N/2 N** | #4 | 2 | 19 | 139 | 77% |
| **TT x 1.5** | **0 N x (TTx1.5)** | #5 | 9 | 48 | 588 | 327% |

Look at the data you recorded in this form to help you in your analysis and your answers. The form is meant to help you in this task.

Add your clear and succinct answers below each question. Answers that are organized and non-rambling will earn more points that those that aren’t. When comparing runtimes, you’ll be referring to percentages generally.

1. Did you get similar results in terms of percentages as show in the example? Why/Why not – what do you think?
2. In the example shown (and perhaps your own), why do you think trials #2 and #3 take more time than #1 – shouldn’t they be close in time? What did your runs show? What do you conclude?
3. In the example, why does trial #4 not take close to half the time of #1? After all you have two threads doing the work of one? ( <- I consider this probably the most important question)
4. In the example, why is trial #5 so much worse?
5. What happens if you comment out the code that executes the .join() calls? Carefully explain. Provide a pair of screenshots that show a simple example (once with and once w/o join()) using three threads. Paste them right below, you’ll be pasting other runs in Part IV.

You must use these pairs: **0 30000 5000 10000 0 20**

Screenshot with join() using 3 threads

Screenshot without join() using 3 threads **repeating above command line arguments**

**Part IV** – Screenshots for experimental runs from table (the source for your table data)

Provide screengrabs of the five required trials *in the order given in the above table*. To earn any credit, they must be readable and show the same values you entered in your table.

Please note that this is *not* testing, that you will have done on your own before.

Trial #1:

A computer screen shot of a computer

Description automatically generated

Trial #2:

A computer screen shot of a computer program

Description automatically generated

Trial #3:

A computer screen shot of a computer program

Description automatically generated

Trial #4:

A computer screen shot of a program

Description automatically generated

Trial #5:

A computer screen with white text

Description automatically generated

Screenshot with join() using 3 threads

A computer screen with white text

Description automatically generated

Screenshot without join() using 3 threads

A computer screen with white text

Description automatically generated

1. If you don’t have access to a personal machine with these specs, plan to do your timed runs on a campus PC. [↑](#footnote-ref-1)
2. It can’t be dulinux – it has to be a single user machine you can control so to eliminate other programs running at the same time that would render your results meaningless. [↑](#footnote-ref-2)
3. I found this value only after several trial runs. It gets pretty close to 3 minutes. Your **N** will most likely be different. [↑](#footnote-ref-3)