This homework assignment contains four word problems and includes a programming task. Be sure to show your work. Correct answers without a show of your approach will receive partial credit only. You may use a calculator, spreadsheets, or you may write a program to help you arrive at the answers. However, the answers to the 4 questions must be yours and yours only. Although you may discuss the homework with your fellow students, you cannot discuss solutions. Please ask Filip if something is not clear.

**Question 1** : (15 points) Assume you are an employee of *We-Tolerate-Errors*, a software company that specializes in writing programs intended to be run concurrently. You are working on a program for which you have the executable but not the source code (you cannot modify the program). All that you’ve been told is that the program has 3 threads:

Thread A: a1 < a2 < a3 < a4  
Thread B: b1 < b2 < b3  
Thread C: c1 < c2

Thread A has 4 instructions, Thread B has 3, and thread C has 2. Each thread’s instructions are executed sequentially (where < specifies the ordering that we’ve seen in lecture). However the three threads are run concurrently, so the orderings among instructions from different threads is unpredictable from one invocation of the program to another. For example the following 2 are instruction histories that might be realized if the program is run concurrently:

History 1: a1 < a2 < b1 < a3 < a4 < c1 < c2 < b2 < b3  
History 1: a1 < b1 < b2 < c1 < c2 < a2 < b3 < a3 < a4

Also assume that dependencies among variables in use by the threads necessitate that the following constraints (instruction histories) be maintained:

**Constraint 1** : a1 < b3  
**Constraint 2** : c2 < a4

*We-Tolerate-Errors* guarantees their software to be correct to a certain fault tolerant level. The question you must answer: **what percentage of the invocations of the software (when run concurrently) will give an incorrect result?** An Incorrect result is achieved when an execution history violates one of the constraints.

**Question 2** : (10 pts) Why do semaphores have no getter methods (strictly speaking, a getter method is a method that ONLY retrieves the value of a variable and which does NOT update the variable’s value)?
**Question 3**: (15 points) You've been given a program whose pseudocode is shown below. The `/beginConcurrent, /endConcurrent` specify the start and end portions of the code that is run concurrently. Add semaphore(s) as needed, and invoke the `wait()` and `signal()` methods as needed, so when run the program outputs the following:

\[ x=13, \ y=11, \ z=24 \]

Assume that the variables \( x, \ y, \) and \( z \) are global variables, and that each of the two statements in Procedure1 and Procedure2 are atomic.

```java
function procedure1(){
   y = x - 3;
   y = x + y;
}

function procedure2(){
   x = z;
   z = x + y;
}

function main(){
   x = 1;
   y = 4;
   z = 13;

   /beginConcurrent
   procedure2();
   procedure1();
   /endConcurrent

   print("x=", x, " y=", y, " z=", z);
}
```

**Question 4**: (10 pts) Assume that you have a computer whose cache access time is 15ns and main memory access time is 150ns. Further assume that you have a computationally intensive program, `quadratics v2.0`, that is crucial to your employer’s business. `Quadratics 2.0` has a cache reuse ratio of 0.7. A new version of that program, `quadratics v3.0`, has just been released. Unfortunately, the cache reuse ratio of the trial version of `quadratics v3.0` when run on your computer is 0.6. Your boss asks you to determine which hardware upgrades to the cache or main memory are needed (the size of main memory and size of cache **must stay the same**) so that `quadratics v3.0` runs just as fast as does `quadratics v2.0`. Please give a quantitative (numerical) answer, and not a qualitative one. In other words, do not just say, “Improve the cache access time.”
**Programming Task** (50 points)

This programming task builds on what you learned in the threading labs. You will write a program to calculate the product of two square matrices. You'll perform the calculation with and without threads, and compare the run-times.

Matrix multiplication is a binary operation that takes as input a pair of matrices, and produces an output, the product, matrix. It is an often-used procedure in many algebra-intensive applications and high-performance programs. For the square matrices A and B shown above, their product is the following:

\[
AB = \begin{pmatrix}
(AB)_{11} & (AB)_{12} & \cdots & (AB)_{1y} \\
(AB)_{21} & (AB)_{22} & \cdots & (AB)_{2y} \\
\vdots & \vdots & \ddots & \vdots \\
(AB)_{x1} & (AB)_{x2} & \cdots & (AB)_{xy}
\end{pmatrix}
\]

where \((AB)_{ij} = \sum_{k=1}^{m} A_{ik}B_{kj}\)

Refer to the lecture slides for a review of matrix multiplication.

**Coding Details**

- Write a function for calculating the product of square matrices, using the naïve (above) method, which has a runtime of \(O(n^3)\), where \(n\) is the dimension of the square matrix.

- When you generate your matrices, they should be composed of doubles or integers. It is up to you whether the matrix values are randomly generated, or hard-coded (for example, matrix A can be composed of all 1s, and matrix B composed of all 2s).

- Use any programming language of your choice. I recommend using C++, for which you’ve been shown (in lecture and in lab), how to implement threads. If you select another language it is up to you to make sure that the language enables threading. Include a README file with detailed compile instructions especially if you are using something other than C++.

- Write a program with a main routine that invokes the matrix multiplication method. Your program should take as input (as command-line arguments is a good idea as I recommend working from the command line on Linux), parameters that a) designate how many times the matrix method should be invoked, b) whether threading should be used (a thread should consist of invoking the matrix multiplication method), and c) how big the matrix should be (see next section). If threading is specified, your program should create a separate thread for each calculation. If threading is NOT specified, then your program should perform the calculations in serial.

**Running your Program (see Figure 1 for pseudocode schematic)**

- Run your program on the Linux account that you have access to.
- Run the program on matrices that are 200 × 200, 400 × 400, 600 × 600, and 800 × 800.
- For each of the 4 sizes of matrices mentioned above, have your program perform matrix multiplication 1, 2, 4, 8, 16 and 32 times, each with (threaded) and without threading (serial).
- Time each run of your program, using the linux/unix `time` command, as was shown in lab or using the `wall` time routine.
Figure 1: Pseudocode for four separate invocations of the program. Note that the `matrixProduct()` invocations for the non-threaded versions could have been invoked using for loops instead of sequentially calling those methods one-by-one. Numerical values of outputs are for demonstration only.

Analyze the run-time

- Record the run-times of your program, and tally the results in a table. For example, the following table has the user run-time values (in seconds) for the threaded invocation of the program. The values in the table are NOT real; they are shown only for demonstration purposes.

<table>
<thead>
<tr>
<th>Matrix Size</th>
<th>1x</th>
<th>2x</th>
<th>4x</th>
<th>8x</th>
<th>16x</th>
<th>32x</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 x 200</td>
<td>0.114</td>
<td>0.114</td>
<td>0.114</td>
<td>0.014</td>
<td>0.171</td>
<td>2.272</td>
</tr>
<tr>
<td>400 x 400</td>
<td>0.930</td>
<td>0.830</td>
<td>0.956</td>
<td>0.963</td>
<td>1.102</td>
<td>1.462</td>
</tr>
<tr>
<td>600 x 600</td>
<td>2.163</td>
<td>2.128</td>
<td>2.294</td>
<td>2.195</td>
<td>2.790</td>
<td>3.339</td>
</tr>
<tr>
<td>800 x 800</td>
<td>7.018</td>
<td>6.715</td>
<td>6.762</td>
<td>6.696</td>
<td>8.605</td>
<td>45.456</td>
</tr>
</tbody>
</table>

- Plot the run-times of your algorithm, for the four matrix sizes, when threading was used, and when it wasn’t. Two sample plots are shown in Figure 2. Be sure to label your axes.

Figure 2: Execution times for threaded (left) and serial (right), for different matrix sizes, for different invocation counts (1x dark blue, 2x orange, 4x yellow, 8x green, 16x purple, 32x light blue).
Writeup

Compose a brief document (2 pages should suffice, even 1 is fine!) that includes:

- Tabulated run-times
- Correctly labeled (axes, titles, units, etc.) plots of run-times
- High-level architecture specs about your computer (number of cores and cache/memory sizes). See Lab 1 for how to gain access to that information.

The document should also include at MOST 2 paragraphs discussing/explaining the following:

1. Does threading improve the performance of your program? If yes, is it reasonable to assume that such a performance gain should always exist, or only under special circumstances? Explain.
2. Is there a thread “limit”, at which point adding more threads does not significantly improve the performance of your program? Regardless of whether you answer yes or no, explain this in the context of the system architecture for the linux account where you ran your program.

Submission

Submit via Canvas:

- A document (.doc, .docx, .pdf) with your answers to the 4 word problems
- A document (.doc, .docx, .pdf) with your answers and explanations to the 2 questions, as well as tabulated run times, plots of run-times, and high level architecture specs, for the programming portion of the assignment
- Source code for your program
- A README file explaining how to compile and run your program.

Rubric

<table>
<thead>
<tr>
<th>Question/Component</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word problems 1-4</td>
<td>50 points</td>
</tr>
<tr>
<td>Two code portion questions</td>
<td>10 points</td>
</tr>
<tr>
<td>Tabulated run-times and correctly labeled plots</td>
<td>15 points</td>
</tr>
<tr>
<td>Source Code, compiles, no errors, threading correctly implemented, etc.</td>
<td>15 points</td>
</tr>
<tr>
<td>README file</td>
<td>10 points</td>
</tr>
<tr>
<td>Total</td>
<td>100 points</td>
</tr>
</tbody>
</table>