**Submission Form**

Name: Siddhesh Dahatonde

Programme: MSc in Computing, Secure Software Engineering

Module Code: CA 670

Assignment Title: Java Threads vs. OpenMP - Matrix Multiplication

Submission Date: 04 / 04 /2022

Module Coordinator: Dr. David Sinclair

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. I have read and understood the Assignment Regulations. I have identified and included the source of all facts, ideas, opinions, and viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the source cited are identified in the assignment references. This assignment, or any part of it, has not been previously submitted by me/us or any other person for assessment on this or any other course of study.

Name: Siddhesh Dahatonde Date: 27 / 02 / 2022

**1. Multithreading in JAVA:**

Multithreading in Java refers to the process of running two or more threads at the same time to maximize CPU utilization.   In Java, multitasking is accomplished by multithreading and multiprocessing, however we prefer multithreading to multiprocessing. This is because the threads share a memory space, which saves memory and makes content swapping between the threads a little faster than the process. Also, a thread is a lightweight process that uses fewer resources to produce and exchange resources.

**2.** **Multithreading in C:**

Multitasking is a better term for multithreading. Multitasking allows you to run two or more programs on your computer at the same time. Multitasking can be done in two ways: process-based and thread-based. Process-based multitasking aids in the execution of multiple programs at the same time. Additionally, thread-based multitasking aids in the handling of multiple sections of the same program running at the same time. Multithreading means that two or more threads are active at the same time. And each thread is in charge of a distinct task. It enables you to carry out multiple tasks at the same time. Multithreaded applications are not supported by C++; instead, they rely fully on the operating system.

**3. OpenMP:**

OpenMP is a library for Symmetric Multi-Processors parallel programming. All threads in an OpenMP software share memory and data. C, C++, and Fortran are all supported by OpenMP. A header file called **omp.h** contains the OpenMP functionalities. The structure of an OpenMP program is that the sections of an OpenMP application are sequential, while others are parallel. An OpenMP program begins with a sequential section that sets up the environment and initializes the variables. When an OpenMP application is launched, it will use one thread in the sequential sections and numerous threads in the parallel sections.

**4. Why OpenMP:**

a. Specify the parallel region

b. Specify whether the variables in the parallel section are private or shared

c. Specify how/if the threads are synchronized

d. Specify how to parallelize loops

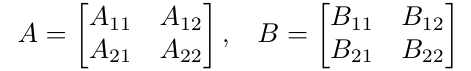
**5. Strassen Algorithm:**

The algorithm which we have used for our implementation is Strassen Algorithm which is an efficient matrix multiplication algorithm. It is faster and has a lower asymptotic complexity than the usual matrix multiplication approach for large matrices, however the naive algorithm is frequently superior for smaller matrices. For extremely large matrices, the Strassen algorithm is slower than the fastest known algorithms, but such algorithms are not helpful in reality because they are significantly slower for matrices of the size which are generally used in today’s world. Strassen's algorithm works for plus/multiply, but not for min-plus or Boolean algebra, where the naive algorithm and combinatorial matrix multiplication work.

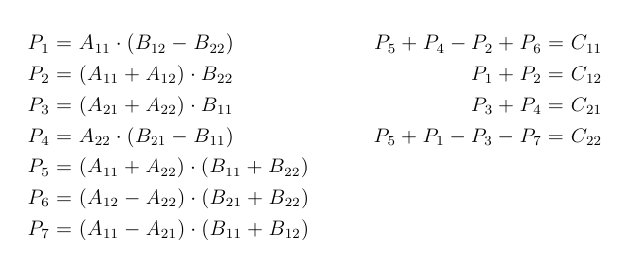
**6. Working of Algorithm:**

a. It is a recursive method for matrix multiplication where we divide the matrix into 4 sub-matrices of dimensions n/2 x n/2 in each recursive step.

b. For example, consider two 4 x 4 matrices A and B that we need to multiply. A 4 x 4 can be divided into four 2 x 2 matrices.



c. Here, Aᵢⱼ and Bᵢⱼ are 2 x 2 matrices and now we can calculate the product of A and B (matrix C) with the following formulas:



d. There are conditions for Strassen’s algorithm to work:

* Both input matrices should be of dimensions n x n.
* n should be a power of 2.

e. If the above conditions are not satisfied, we must pad the matrices with 0 to satisfy the above conditions.

**7. Implementation and Testing:**

To implement the Strassen Algorithm, we need to multiply the whole matrices and for that we must call the StrassenMultiply function recursively by dividing the matrices into four matrices of dimension n/2 x n/2 and calculating the product using the methods described above.

We have developed two programs in Java where we compare the normal multiplication matrix with the Strassen Algorithm and observe the timings.

Please Enter the Number

16

0.4259131 Seconds

Please Enter the Number

32

2.5801773 Seconds

Please Enter the Number

64

19.7131454 Seconds

Please Enter the Number

128

143.600905 Seconds

After running the codes based on the timings, we can see that, after normal execution of the program, it gives better result than the Strassen Algorithm. However, if we run the method on larger inputs, which would take a lot longer, the Strassen algorithm performs better. The experiment we performed also states that the parallel execution is much faster and reliable than the serial implementation.

As we know that the C is closer to the machine code, we will now implement the matrix multiplication in OpenMP and then compare it with the Strassen Algorithm.

Normal Multiplication:

1024

4.4215895

2048

134.7845247

Strassen:

1024

3.8459644

2048

74.5471512

As can be seen, the timing for Strassen improves as the input size grows, and the timing for C is better than java.

**8. Conclusion:**

We observed that the behaviour of the algorithms was different in java and C. Strassen was fastest in C whereas the normal matrix multiplication was fastest in Java. This was because when compared to other programming languages such as C and C++, Java is a safer option. For example, in any matrix application, the indexing is checked and calculated every time, whereas C does not. As a result, when one chooses to play it safe by utilizing Java, performance suffers. Because Java has a long start-up time, short-running operations are likely to be completed before the JVM has even started up. The use of Java threads should be considered a calculated risk. Despite the fact that the 'thread' component helps reduce processing time by breaking down a large problem into smaller subproblems and tackling them all at once, Java threads can occasionally produce more overhead than benefit. Allocation and initialization of a huge block of memory for the thread stack, system calls to create or register native threads with the host OS, and creation, initialization, and adding of descriptors to JVM data structures are all costs associated with Java Thread-Creation. In most circumstances, the thread also holds all of the resources involved for as long as it is alive, which is a drawback.

**References**

1. https://tildesites.bowdoin.edu/~ltoma/teaching/cs3225-GIS/fall17/Lectures/openmp.html

2. https://akarshbolar.wordpress.com/2017/04/02/matrix-multiplication-in-java/

3. https://martin-thoma.com/matrix-multiplication-python-java-cpp/#tocAnchor-1-10-4

4. https://martin-thoma.com/matrix-multiplication-python-java-cpp/#The\_Tests

5. https://github.com/ilya1032/Multithread-course-work-Java-threads-pthreads-OpenMP-

5. https://rohankanuri.wordpress.com/2017/03/11/strassens-matrix-multiplication/

6. https://github.com/Shafaet/OpenMP-Examples

7. https://www.sanfoundry.com/java-program-strassen-algorithm/

8. https://www.codesdope.com/blog/article/strassens-matrix-multiplication/