Name: Aniket Suryakant Angwalkar

Student ID Number: 23269100

Email: aniket.angwalkar2@mail.dcu.ie

Program of Study: M.Sc. in Computing (Secure Software Engineering)

Module Code: CA670

Module Name: Concurrent Programming

Date of Submission: 08/04/2023

I understand that the University regards breaches of academic integrity and plagiarism as grave and serious. I have read and understood the DCU Academic Integrity and Plagiarism Policy. I accept the penalties that may be imposed should I engage in practice or practices that breach this policy. I have identified and included the source of all facts, ideas, opinions, viewpoints of others in the assignment references. Direct quotations, paraphrasing, discussion of ideas from books, journal articles, internet sources, module text, or any othe source whatsoever are acknowledged, and the sources cited are identified in the assignment references. Any use of generative AI or search will be described in a one-page appendix including prompt queries.

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. By signing this form or by submitting this material online I confirm that this assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study. By signing this form or by submitting material for assessment online I confirm that I have read and understood DCU Academic Integrity and Plagiarism Policy.

Signed Name: Aniket Suryakant Angwalkar

Date: 08/04/2024

ASSIGNMENT 2

Design of the Matrix Multiplication Program in Java:

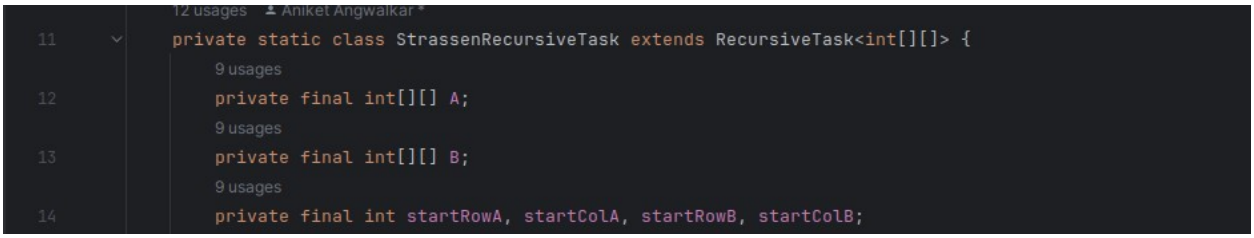
For the program and creating an efficient solution to the Matrix Multipllication problem, I've utilized the Strassen Algorithm for Matrix multiplication. The program takes user input for the size of the matrices with a maximum size limited to 4096 x 4096. This algorithm is efficient in itself since it utilizes the divide and conquer approach from dynamic programming and reduces the multiplication steps of reduced matrices to 7 steps instead of 8. In order to add greater efficiency, I have added a multi-threaded multi-core approach to the reduced matrix multiplication steps.

The program for this solution has been created into one Java Class, and another class nested into the major class as follows:

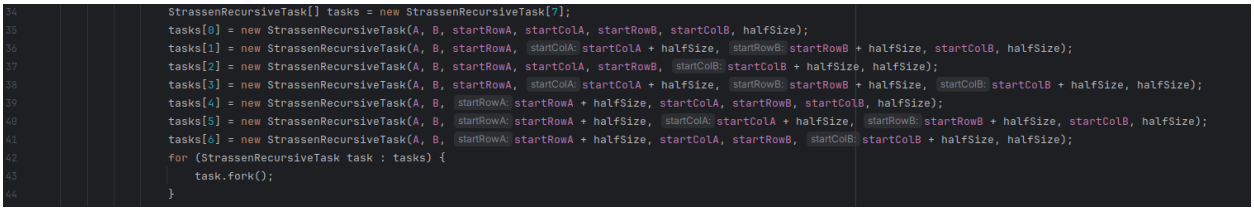
1. ThreadedStrassen
2. StrassesRecursiveTask

StrassesRecursiveTask:

- The StrassenRecursiveTask extends the RecursiveTask api from java.util.concurrent. This class handles the major part of the calculation process that computes the multiplied matrices to be merged and calculated.



- This class consists of the compute method that creates 7 tasks for the Strassen Algorithms' 7 steps and then joins then matrices.

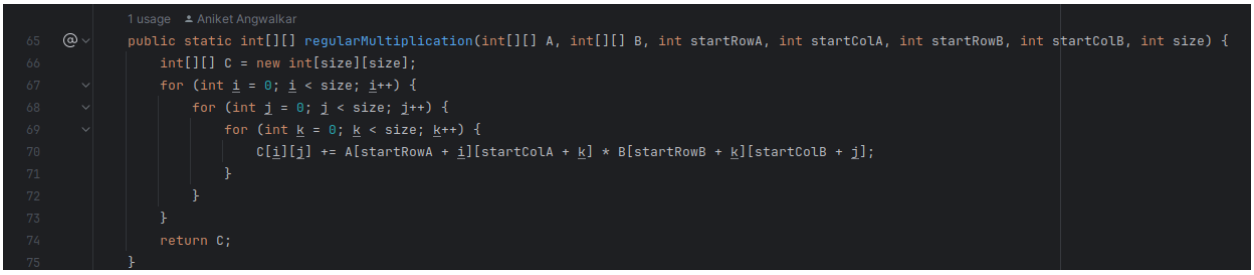


ThreadedStrassen:

- The ThreadedStrassen class is the main major class of the program which also contains the main class.

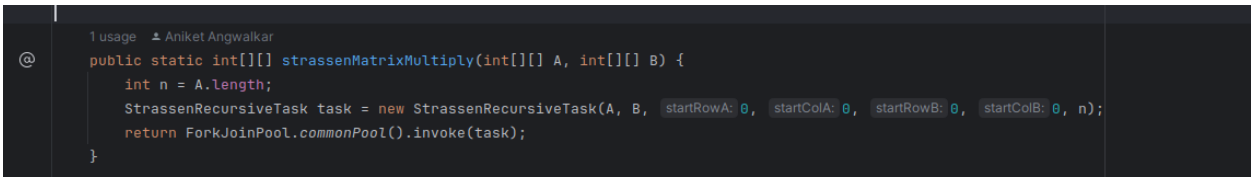
- This class consists of other utility methods such as joinMatrices, addMatrices, subMatrices, necessary for the implementation of the program.

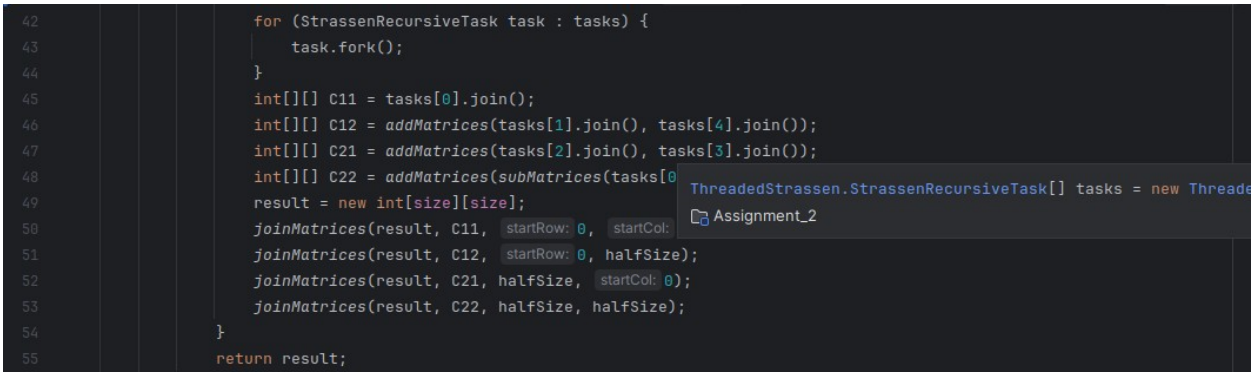
- This class also consists an extra utility method for cases where the matrix size is less than 128 x 128 to utilize regular matrix multiplication.



- I have also created one generateRandomMatrix method to create matrices with random values between 0 and 50.

- This class utilizes the ForkJoinPool methods for Thread manipulation, and manages worker threads and provides a deck (double ended queue) that stores and executes tasks in the strassenMatrixMultiply method. The ForkJoinPool api is constructed with a given target parellelism level by default, equal to the number of cores. In this case, my device incorporates 16 cores by design and it's same to assume that the program will make use of the 16 in conjunction with the task distribution.





Design of the Matrix Multiplication Program in C using OpenMP:

We use a regular matrix multiplication method in this program alongside the OpenMP implementation for the nested looping mechanism. The program takes user input for the size of the matrices similar to java with a maximum size limited to 4096 x 4096. The program has 4 methods including the main method. The methods are main, generateRandomMatrix, matrixMultiply and printMatrix.

The main method is the utility program method that executes all the methods, generates the random matrices to be multiplied and then passes them on to the matrixMultiply method. This method is also responsible for allocating memory for the program as well as freeing the memory at the end to maintain consistency of the program.

```c
int main() {
    int rows = 2048, cols = 2048;

    printf("Enter dimensions of the first matrix (rows cols): ");
    scanf("%d %d", &rows, &cols);


    if (rows > 4096 || cols > 4096) {
        printf("Error: Matrix dimensions exceed the limit of 4096x4096.\n");
        return 1;
    }

    int **matrix1 = (int **)malloc(rows * sizeof(int *));
    int **matrix2 = (int **)malloc(rows * sizeof(int *));
    int **result = (int **)malloc(rows * sizeof(int *));
    for (int i = 0; i < rows; i++) {
        matrix1[i] = (int *)malloc(cols * sizeof(int));
        result[i] = (int *)malloc(cols * sizeof(int));      You, 3 days ago • Added Multithreaded C and Java Implementati
    }
    for (int i = 0; i < rows; i++) {
        matrix2[i] = (int *)malloc(cols * sizeof(int));
    }
```

```c
    printf("Execution Time: %.2f milliseconds\n", exec_time);


    for (int i = 0; i < rows; i++) {
        free(matrix1[i]);
        free(result[i]);
    }
    for (int i = 0; i < rows; i++) {
        free(matrix2[i]);
    }
    free(matrix1);
    free(matrix2);
    free(result);

    return 0;
}
```

The matrixMultiply method makes use of the OpenMP #pragma annotation for implementation of the parellel for loop with the collapse(2) tag. The collapse annotation makes sure the program is well distributed in terms of load and since the nested loops do not depend on the outer loops, the program behaviour is not unprecedented.

```
24  double matrixMultiply(int rows, int cols, int **matrix1, int **matrix2,
25                        int **result) {
26
27      double start_time = omp_get_wtime();
28      #pragma omp parallel for collapse(2)
29      for (int i = 0; i < rows; i++) {
30          for (int j = 0; j < cols; j++) {
31              result[i][j] = 0;
32              for (int k = 0; k < cols; k++) {
33                  result[i][j] += matrix1[i][k] * matrix2[k][j];
34              }
35          }
36      }
37      double end_time = omp_get_wtime();
38      return (end_time - start_time) * 1000.0;
39  }
```

Design of the Result plotting and profiling using Python and matplotlib:

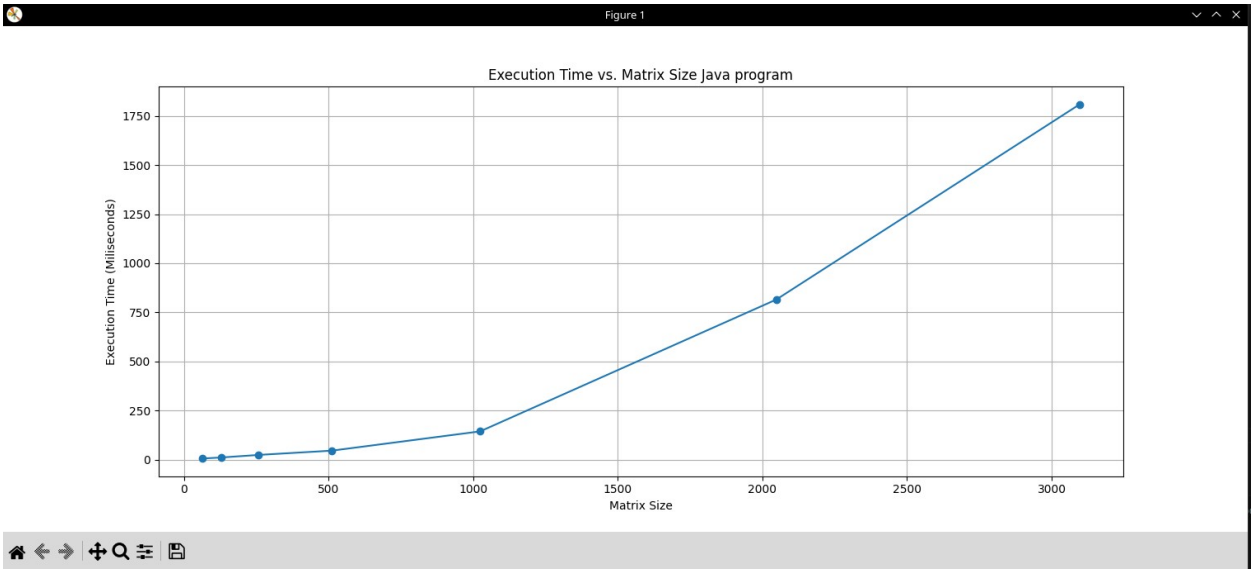
This program executes by taking user input so as to decide which program to execute. The two options are java and c. The program executes a set of matrix multiplication scenarios with sizes ranging from 64 x 64 to 3096 x 3096.

```
    return float(time_str)

matrix_list = [64, 128, 256, 512, 1024, 2048, 3096]

prog = 0
if len(sys.argv) > 1:
    prog = sys.argv[1]
```

Execution Outputs

Java Execution Plot:

```
❯ ls
 CA670_Assignment_2.docx   fork_join.png   java_exec_time.png   plot_size.png   regular.png   'ThreadedStrassen$StrassenRecursiveTask.class'
 c_exec_time.png           fork_pool.png   malloc.png           pragma.png      Strassen       ThreadedStrassen.class
 fork_2.png                free.png        plot.py              recursive.png   Strassen.c     ThreadedStrassen.java
❯ python plot.py java
6
11
24
46
144
815
1808
```



C Execution Plot:

```
1808
❯ python plot.py c
0.19
0.60
5.38
48.89
386.72
3495.46
12721.37
    ~/Documents/Coursework/Concurrent Programming/Assignments/Assignment_2/src | master +4 !6 ?16 ·····
```