Multicore computing

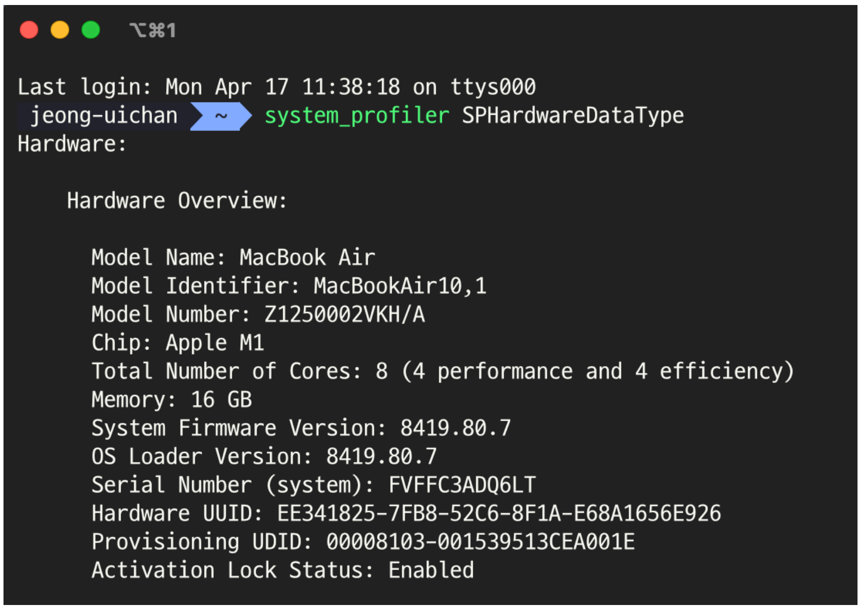
Project #1 – problem2

Department : Software

Name : Jeong Eui Chan

Student ID : 20195914

MY computer system enviromnent



```
⌥⌘1

Last login: Mon Apr 17 11:38:18 on ttys000
 jeong-uichan  ~  system_profiler SPHardwareDataType
Hardware:

    Hardware Overview:

      Model Name: MacBook Air
      Model Identifier: MacBookAir10,1
      Model Number: Z1250002VKH/A
      Chip: Apple M1
      Total Number of Cores: 8 (4 performance and 4 efficiency)
      Memory: 16 GB
      System Firmware Version: 8419.80.7
      OS Loader Version: 8419.80.7
      Serial Number (system): FVFFC3ADQ6LT
      Hardware UUID: EE341825-7FB8-52C6-8F1A-E68A1656E926
      Provisioning UDID: 00008103-001539513CEA001E
      Activation Lock Status: Enabled
```

Both Problem1 and Problem2 were performed in an environment with the following specifications:
The cpu of my computer uses an Apple M1 chip with 8 cores, consisting of 4 P-cores and 4 E-cores and The maximum clock speed of my cpu is 3.2GH.
The memory capacity is 16GB, and the memory type used is LPDDR4. The operating system being used is macOS Ventrura version 13.2.1.

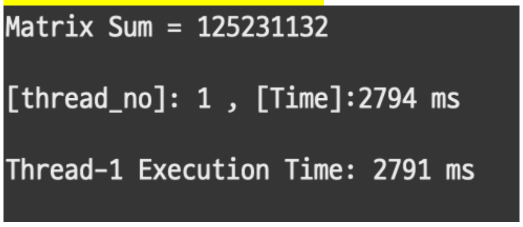
1. parallel MatmultD

1-1) How to compile & execute

To create the source code, first create a project in Eclipse and then create a project file called "Thread test". Next I created a package called "Problem2" and then a class called "parallelMatmultD". After writing the code in this class, designating a thread for compilation and execution, I input the mat500.txt file and execute it.

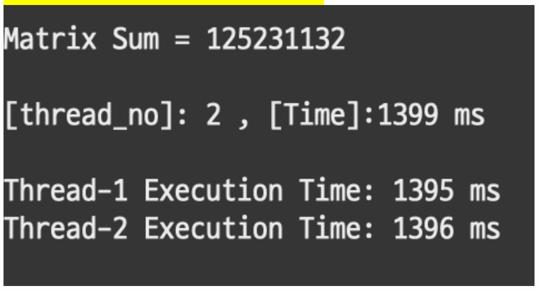
1-2) Result screen

[Entire Threads : 1]



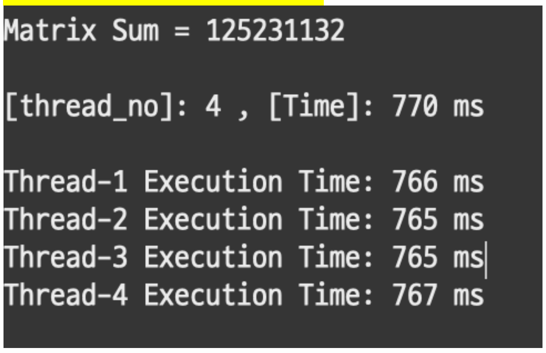
```
Matrix Sum = 125231132

[thread_no]: 1 , [Time]:2794 ms

Thread-1 Execution Time: 2791 ms
```

[Entire Threads : 2]



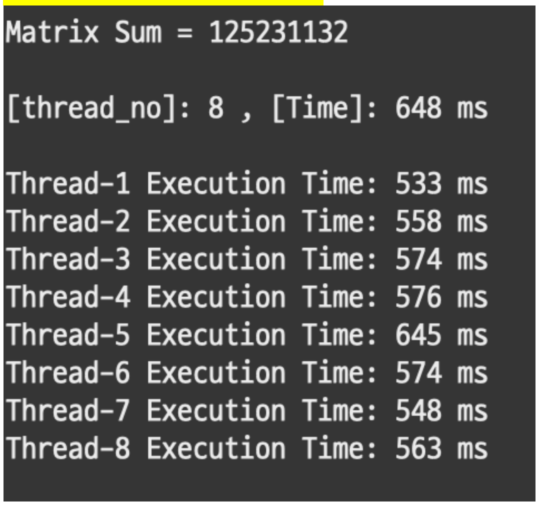
```
Matrix Sum = 125231132

[thread_no]: 2 , [Time]:1399 ms

Thread-1 Execution Time: 1395 ms
Thread-2 Execution Time: 1396 ms
```

[Entire Threads : 4]



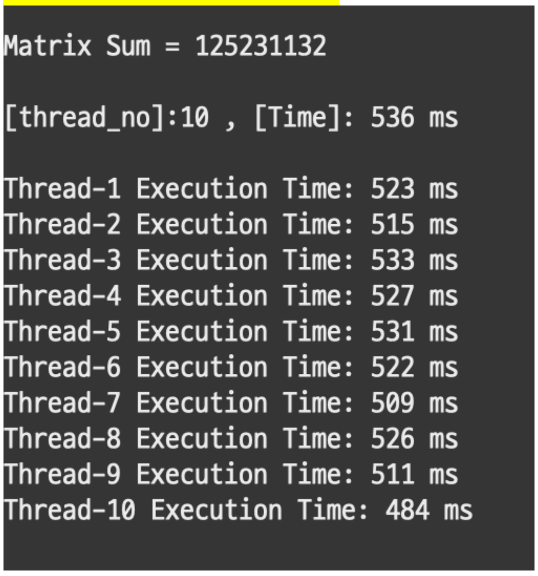
```
Matrix Sum = 125231132

[thread_no]: 4 , [Time]: 770 ms

Thread-1 Execution Time: 766 ms
Thread-2 Execution Time: 765 ms
Thread-3 Execution Time: 765 ms
Thread-4 Execution Time: 767 ms
```

[Entire Threads : 8]



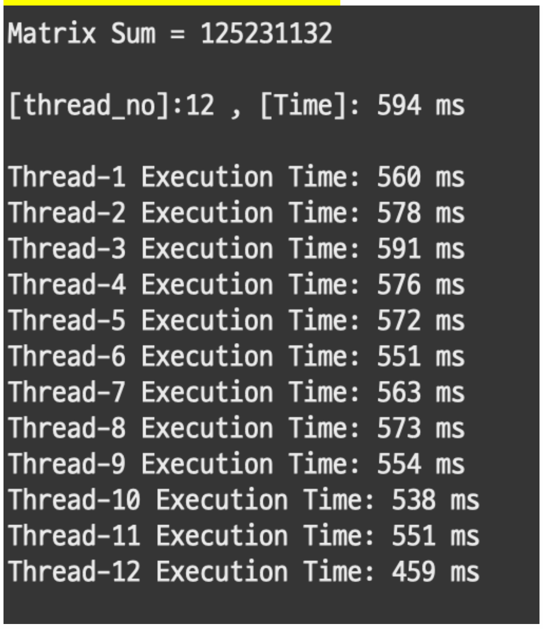
```
Matrix Sum = 125231132

[thread_no]: 8 , [Time]: 648 ms

Thread-1 Execution Time: 533 ms
Thread-2 Execution Time: 558 ms
Thread-3 Execution Time: 574 ms
Thread-4 Execution Time: 576 ms
Thread-5 Execution Time: 645 ms
Thread-6 Execution Time: 574 ms
Thread-7 Execution Time: 548 ms
Thread-8 Execution Time: 563 ms
```

[Entire Threads : 10]



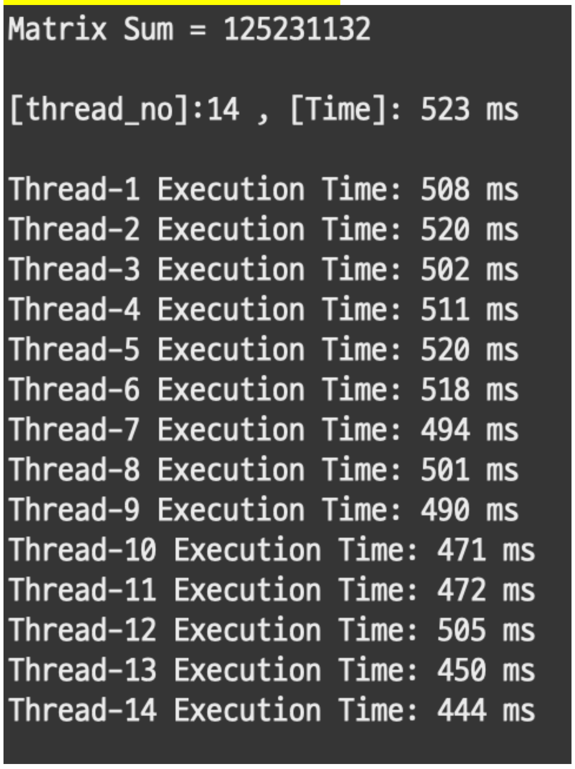
```
Matrix Sum = 125231132

[thread_no]:10 , [Time]: 536 ms

Thread-1 Execution Time: 523 ms
Thread-2 Execution Time: 515 ms
Thread-3 Execution Time: 533 ms
Thread-4 Execution Time: 527 ms
Thread-5 Execution Time: 531 ms
Thread-6 Execution Time: 522 ms
Thread-7 Execution Time: 509 ms
Thread-8 Execution Time: 526 ms
Thread-9 Execution Time: 511 ms
Thread-10 Execution Time: 484 ms
```

[Entire Threads : 12]



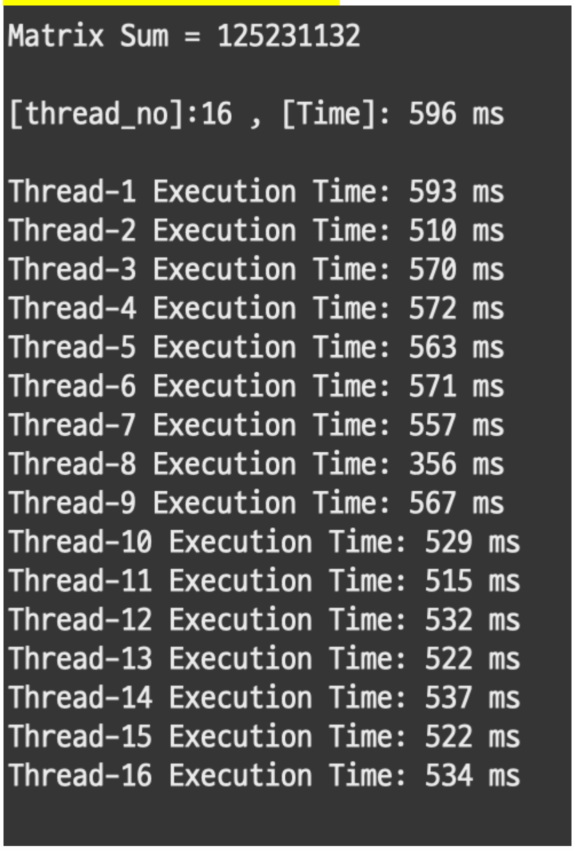
```
Matrix Sum = 125231132

[thread_no]:12 , [Time]: 594 ms

Thread-1 Execution Time: 560 ms
Thread-2 Execution Time: 578 ms
Thread-3 Execution Time: 591 ms
Thread-4 Execution Time: 576 ms
Thread-5 Execution Time: 572 ms
Thread-6 Execution Time: 551 ms
Thread-7 Execution Time: 563 ms
Thread-8 Execution Time: 573 ms
Thread-9 Execution Time: 554 ms
Thread-10 Execution Time: 538 ms
Thread-11 Execution Time: 551 ms
Thread-12 Execution Time: 459 ms
```

[Entire Threads : 14]



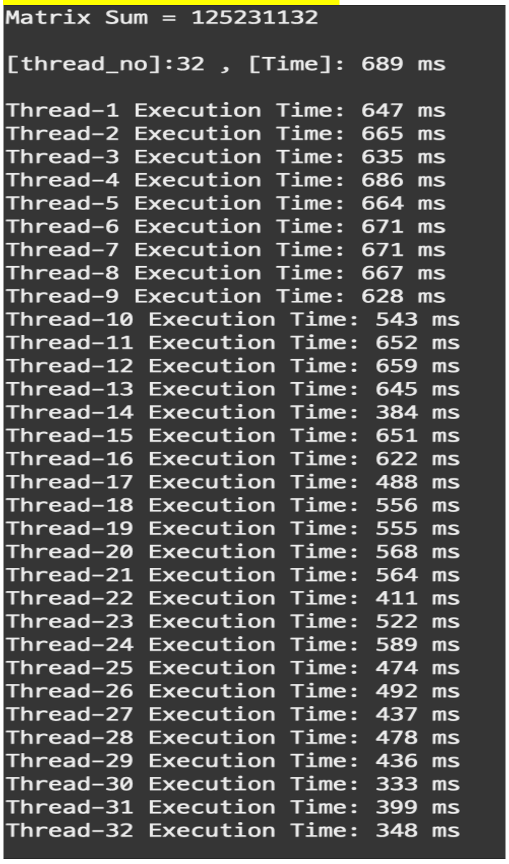
```
Matrix Sum = 125231132

[thread_no]:14 , [Time]: 523 ms

Thread-1 Execution Time: 508 ms
Thread-2 Execution Time: 520 ms
Thread-3 Execution Time: 502 ms
Thread-4 Execution Time: 511 ms
Thread-5 Execution Time: 520 ms
Thread-6 Execution Time: 518 ms
Thread-7 Execution Time: 494 ms
Thread-8 Execution Time: 501 ms
Thread-9 Execution Time: 490 ms
Thread-10 Execution Time: 471 ms
Thread-11 Execution Time: 472 ms
Thread-12 Execution Time: 505 ms
Thread-13 Execution Time: 450 ms
Thread-14 Execution Time: 444 ms
```

[Entire Threads : 16]



```
Matrix Sum = 125231132

[thread_no]:16 , [Time]: 596 ms

Thread-1 Execution Time: 593 ms
Thread-2 Execution Time: 510 ms
Thread-3 Execution Time: 570 ms
Thread-4 Execution Time: 572 ms
Thread-5 Execution Time: 563 ms
Thread-6 Execution Time: 571 ms
Thread-7 Execution Time: 557 ms
Thread-8 Execution Time: 356 ms
Thread-9 Execution Time: 567 ms
Thread-10 Execution Time: 529 ms
Thread-11 Execution Time: 515 ms
Thread-12 Execution Time: 532 ms
Thread-13 Execution Time: 522 ms
Thread-14 Execution Time: 537 ms
Thread-15 Execution Time: 522 ms
Thread-16 Execution Time: 534 ms
```

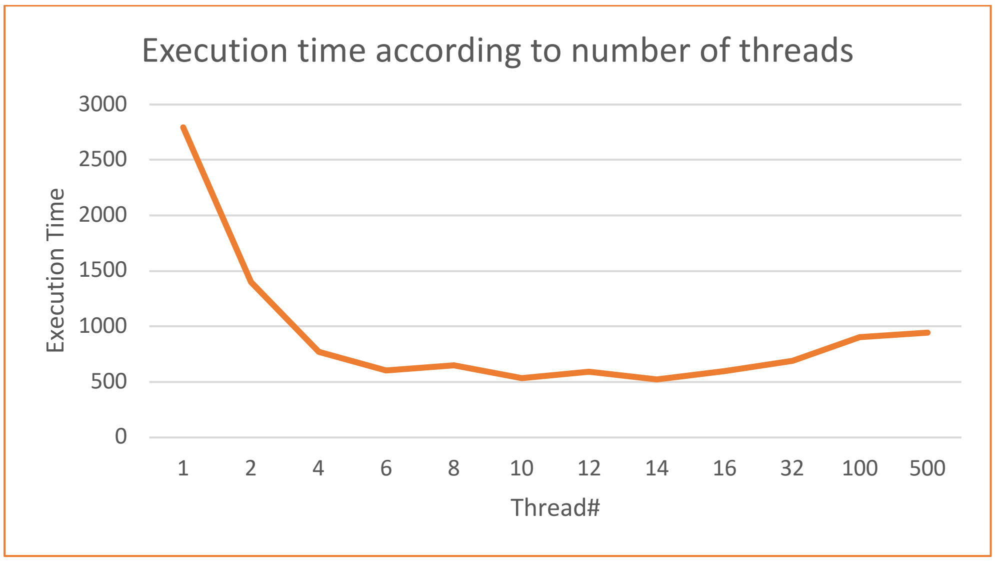
[Entire Threads : 32]



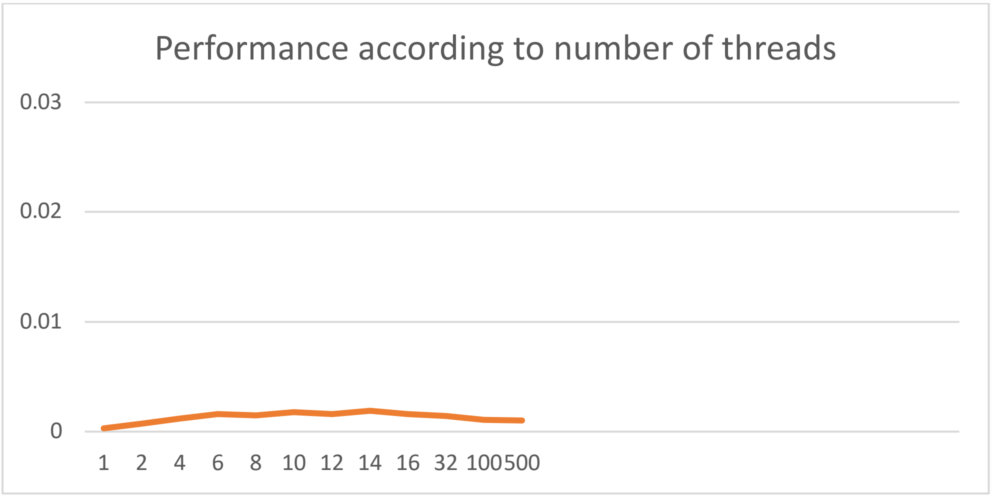
```
Matrix Sum = 125231132

[thread_no]:32 , [Time]: 689 ms

Thread-1 Execution Time: 647 ms
Thread-2 Execution Time: 665 ms
Thread-3 Execution Time: 635 ms
Thread-4 Execution Time: 686 ms
Thread-5 Execution Time: 664 ms
Thread-6 Execution Time: 671 ms
Thread-7 Execution Time: 671 ms
Thread-8 Execution Time: 667 ms
Thread-9 Execution Time: 628 ms
Thread-10 Execution Time: 543 ms
Thread-11 Execution Time: 652 ms
Thread-12 Execution Time: 659 ms
Thread-13 Execution Time: 645 ms
Thread-14 Execution Time: 384 ms
Thread-15 Execution Time: 651 ms
Thread-16 Execution Time: 622 ms
Thread-17 Execution Time: 488 ms
Thread-18 Execution Time: 556 ms
Thread-19 Execution Time: 555 ms
Thread-20 Execution Time: 568 ms
Thread-21 Execution Time: 564 ms
Thread-22 Execution Time: 411 ms
Thread-23 Execution Time: 522 ms
Thread-24 Execution Time: 589 ms
Thread-25 Execution Time: 474 ms
Thread-26 Execution Time: 492 ms
Thread-27 Execution Time: 437 ms
Thread-28 Execution Time: 478 ms
Thread-29 Execution Time: 436 ms
Thread-30 Execution Time: 333 ms
Thread-31 Execution Time: 399 ms
Thread-32 Execution Time: 348 ms
```

4. table and graph

Exec Time	1	2	4	6	8	10
Static(block)	2794ms	1399ms	770ms	602ms	648ms	536ms
Exec Time	12	14	16	32	100	500
Static(block)	594ms	523ms	596ms	689ms	903ms	946ms



Performance	1	2	4	6	8	10
Static(block)	0.02ms	0.03ms	0.14ms	0.14ms	0.14ms	0.14ms
Performance	1	2	4	6	8	100
Static(block)	0.02ms	0.03ms	0.14ms	0.14ms	0.14ms	0.14ms



5. explain/analysis on the result and why such result are obtained

The program performs parallel matrix multiplication using multithreading. The number of threads used for computation can be determined by the user or by using the default value of 1. As the number of threads increases, the work is divided among multiple threads, which results in improved execution time.

However, as the number of threads increases, the execution time decreases and eventually reaches a similar result time. When the number of threads is increased to 100 and 500, the execution time increases again. This means that despite increasing the number of threads, there is no improvement in speed. The work is statically distributed among threads for matrix multiplication. Each thread is assigned a specific range of rows to calculate. The work distribution may not be uniform, causing some threads to complete their work earlier while others take longer. Due to such reasons, the overall speed improvement may be limited as the number of threads increases. Another reason to consider is the overhead problem as discussed in Problem1.

JAVA source code of parallelMatmultD

```java
package Problem2;

import java.util.*;
import java.lang.*;

public class parallelMatmultD {
      private static Scanner sc = new Scanner(System.in);

      private static MatrixMultiplier[] matManager;

    public static void main(String[] args) {


        int thread_no = (args.length == 1) ?
Integer.parseInt(args[0]) : 1;

        int[][] a = readMatrix();
        int[][] b = readMatrix();


        long startTime = System.currentTimeMillis();
        int[][] c = multMatrix(a, b, thread_no);
        long endTime = System.currentTimeMillis();


        printMatrix(c);


        System.out.printf("[thread_no]:%2d , [Time]:%4d ms\n",
thread_no, endTime - startTime);
        System.out.println();

        for (int i = 0; i < thread_no; i++) {
            System.out.printf("Thread-%d Execution Time: %d
ms\n", i + 1, matManager[i].returnTime());
        }
    }

    public static int[][] readMatrix() {
        int rows = sc.nextInt();
        int cols = sc.nextInt();
        int[][] result = new int[rows][cols];
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                result[i][j] = sc.nextInt();
            }
        }
        return result;
    }

    public static void printMatrix(int[][] mat) {
        int rows = mat.length;
        int columns = mat[0].length;
        int sum = 0;
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < columns; j++) {
                System.out.printf("%4d ", mat[i][j]);
                sum += mat[i][j];
            }
            System.out.println();
        }
        System.out.println();
        System.out.println("Matrix Sum = " + sum + "\n");
    }


    public static int[][] multMatrix(int[][] a, int[][] b, int
thread_no) {
        int rows = a.length;
        int cols = b[0].length;


        if (rows == 0 || a[0].length != b.length) {
            return new int[0][0];
        }

        int[][] result = new int[rows][cols];
        matManager = new MatrixMultiplier[thread_no];
        Thread[] threads = new Thread[thread_no];



        for (int i = 0; i < thread_no; i++) {

           matManager[i] = new MatrixMultiplier(a, b, result,
i * rows / thread_no, (i + 1) * rows / thread_no);
            threads[i] = new Thread(matManager[i]);
            threads[i].start();
        }

        for (int i = 0; i < thread_no; i++) {
            try {
                threads[i].join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        return result;
    }
    private static class MatrixMultiplier implements Runnable
{
        private int[][] a, b, result;
        private int rowStart, rowEnd;
        private long executionTime;

        public MatrixMultiplier(int[][] a, int[][] b, int[][]
result, int rowStart, int rowEnd) {
            this.a = a;
            this.b = b;
            this.result = result;
            this.rowStart = rowStart;
            this.rowEnd = rowEnd;
        }

        public void run() {
            long startTime = System.currentTimeMillis();
            for (int i = rowStart; i < rowEnd; i++) {
                for (int j = 0; j < b[0].length; j++) {
                    for (int k = 0; k < a[0].length; k++) {
                      synchronized(this) {
                        this.result[i][j] += a[i][k] *
b[k][j];
                    }
                      }

                }
            }
            long endTime = System.currentTimeMillis();
            executionTime = endTime - startTime;
        }

        public long returnTime() {
            return executionTime;
        }
    }
}
```