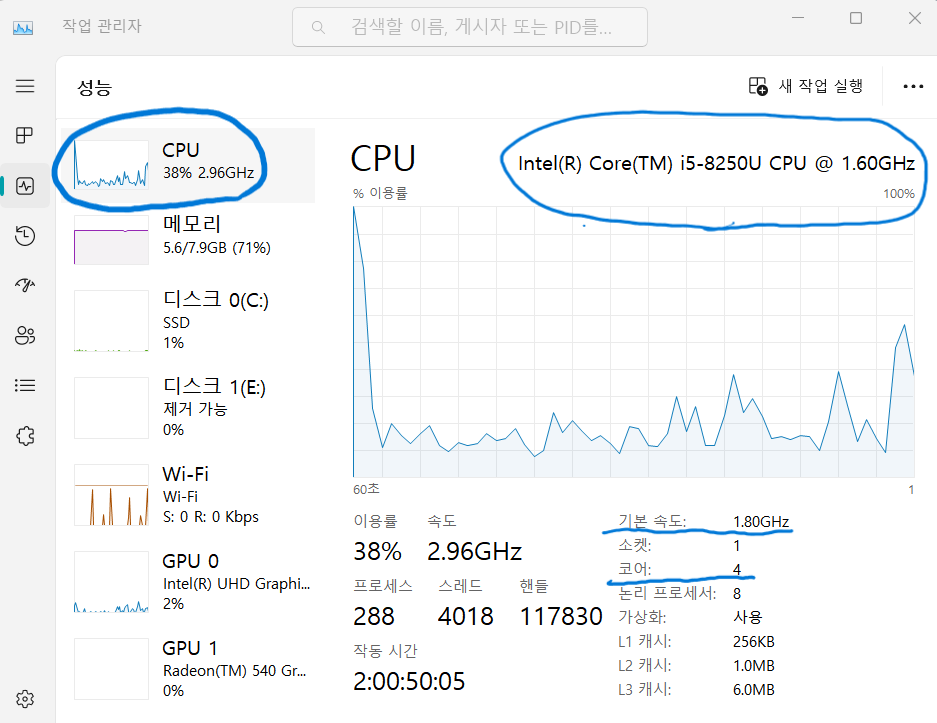
|  |  |
| --- | --- |
|  |  |
| Problem2 Report |  |
|  |  |
|  | Student NO: 20183784Student Name: 노현진 |
|  |  |

**[Environment]**

* **CPU**

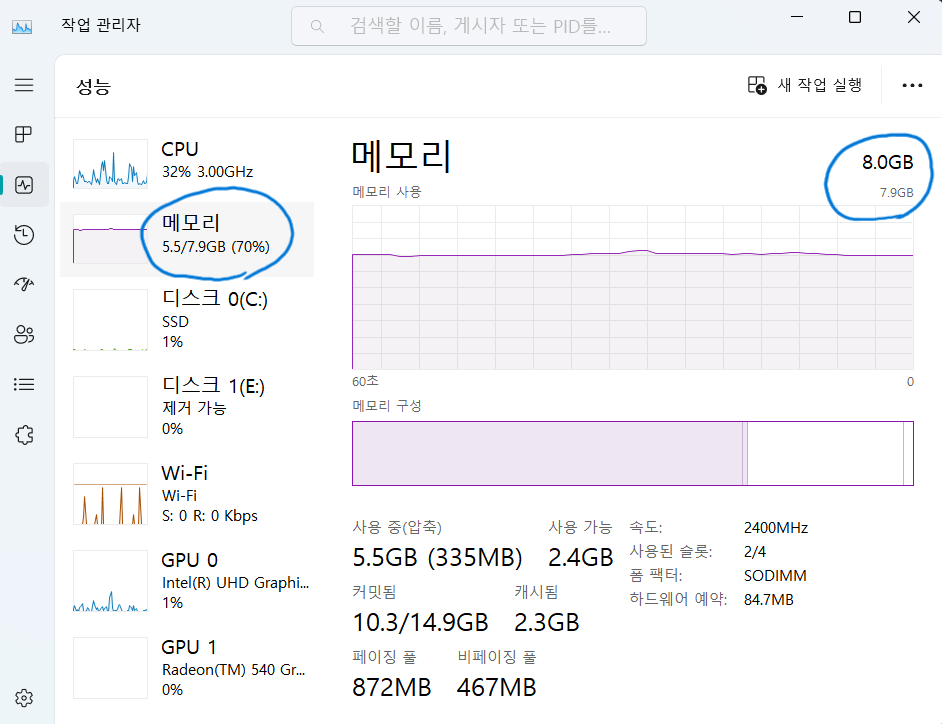


**CPU type: Intel® Core™ i5-8250U CPU**

**Clock Speed: 1.80GHz**

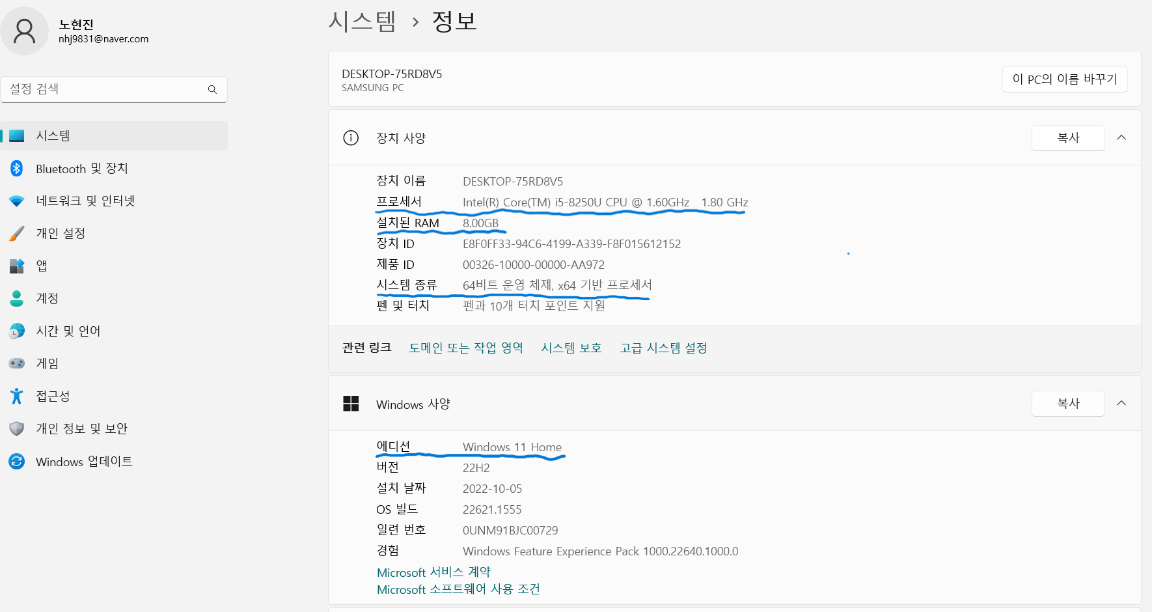
**Number of cores: 4**

* **Memory**



**Memory size: 8.0GB**

* **OS**



**OS type: Windows 11**

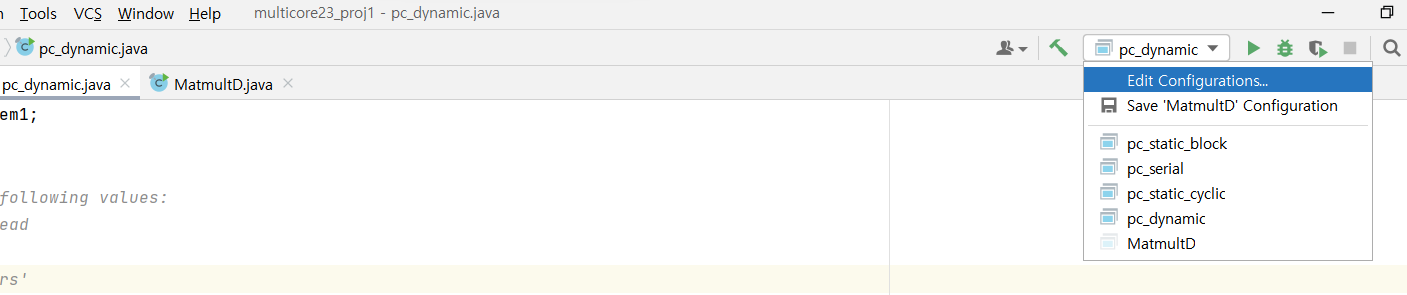
**[Source Code]**

* **MatmultD.java**

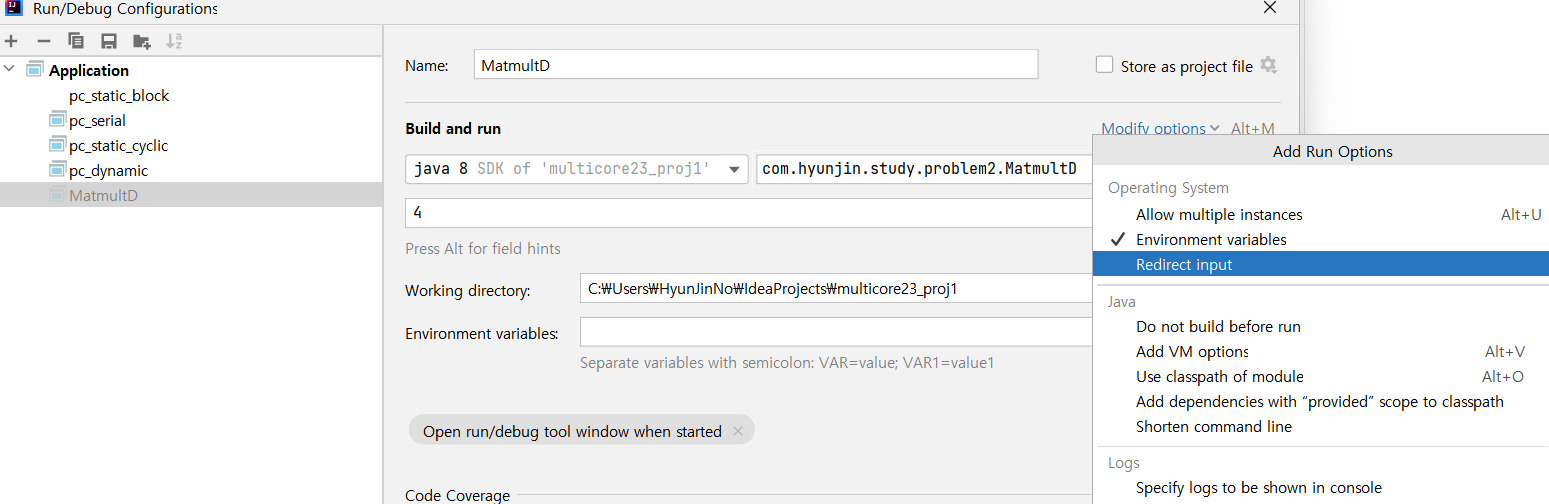
package com.hyunjin.study.problem2;  
  
import java.util.\*;  
import java.lang.\*;  
  
*/\*  
 This program should print the following values:  
 (1) execution time of each thread  
 (2) execution time when using all threads  
 (3) sum of all elements in the resulting matrix  
\*/*public final class MatmultD {  
 private static final Scanner *sc* = new Scanner(System.*in*);  
  
 public static void main(String[] args) {  
 int thread\_no = 1;  
 if (args.length == 1) {  
 thread\_no = Integer.*parseInt*(args[0]); *// Possible number of threads: 1, 2, 4, 6, 8, 10, 12, 14, 16, 32* }  
  
 int[][] a = *readMatrix*();  
 int[][] b = *readMatrix*();  
  
 if (a.length == 0) {  
 System.*out*.println("Invalid Input.");  
 return;  
 }  
 if (a[0].length != b.length) {  
 System.*out*.println("Invalid Dimension.");  
 return;  
 }  
  
 int[][] c = new int[a.length][b[0].length];  
 MatrixThread[] matrixThreads = new MatrixThread[thread\_no];  
  
 int endRow = a.length;  
 int cycle = thread\_no;  
 for (int i = 1; i <= thread\_no; i++) {  
 int startRow = i - 1;  
 matrixThreads[i - 1] = new MatrixThread(a, b, c, startRow, endRow, cycle);  
 }  
  
 long startTime = System.*currentTimeMillis*();  
  
 for (MatrixThread matrixThread : matrixThreads) {  
 matrixThread.start();  
 }  
  
 try {  
 for (MatrixThread matrixThread : matrixThreads) {  
 matrixThread.join();  
 }  
 } catch (Exception ignored) {}  
  
 long endTime = System.*currentTimeMillis*();  
 long timeDiff = endTime - startTime;  
  
 *printMatrix*(c);  
 System.*out*.printf("\n[thread\_no]:%2d , [Time]:%4d ms\n", thread\_no, timeDiff);  
 }  
  
 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) {  
 System.*out*.println("\nMatrix[" + mat.length + "][" + mat[0].length + "]");  
 int cols = mat[0].length;  
 int sum = 0;  
 for (int[] ints : mat) {  
 for (int j = 0; j < cols; j++) {  
 *//System.out.printf("%4d " , ints[j]);* sum += ints[j];  
 }  
 *//System.out.println();* }  
 *//System.out.println();* System.*out*.println("Matrix Sum = " + sum + "\n");  
 }  
}  
  
class MatrixThread extends Thread {  
 private final int[][] a; *// a[m][n]* private final int[][] b; *// b[n][p]* private final int[][] c; *// c[m][p]* private final int startRow;  
 private final int endRow;  
 private final int cycle;  
  
 public MatrixThread(int[][] a, int[][] b, int[][] c, int startRow, int endRow, int cycle) {  
 this.a = a;  
 this.b = b;  
 this.c = c;  
 this.startRow = startRow;  
 this.endRow = endRow;  
 this.cycle = cycle;  
 }  
  
 @Override  
 public void run() {  
 long startTime = System.*currentTimeMillis*();  
  
 for (int i = startRow; i < endRow; i += cycle) {  
 for (int j = 0; j < c[0].length; j++) {  
 for (int k = 0; k < a[0].length; k++) {  
 c[i][j] += a[i][k] \* b[k][j];  
 }  
 }  
 }  
  
 long endTime = System.*currentTimeMillis*();  
 long timeDiff = endTime - startTime;  
 System.*out*.println(this.getName() + " ==> Execution Time: " + timeDiff + "ms");  
 }  
}

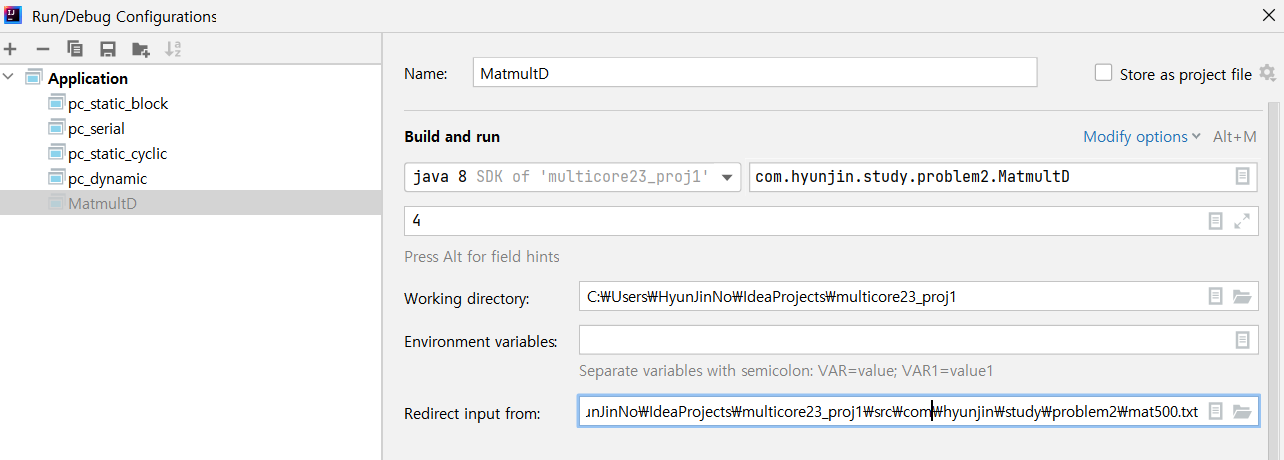
* **How to compile and execute**

1. **Firstly, install Intellij IDEA.**
2. **After installation, open the submitted file.**
3. **Before execution, click “Edit Configurations..”**

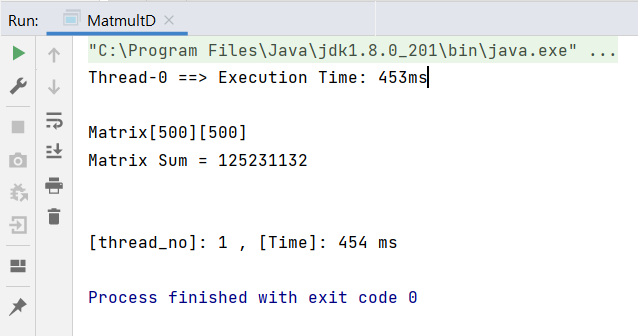
****

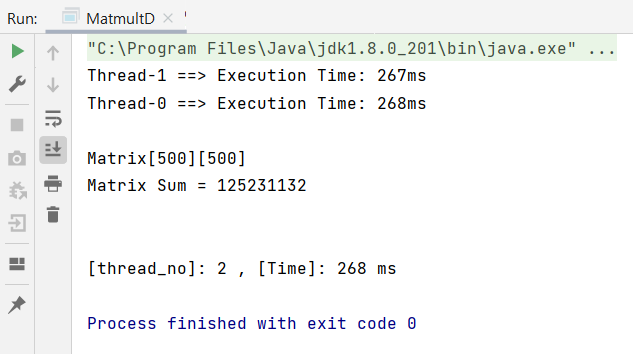
1. **Click “Modify Options” > “Redirect Input”**

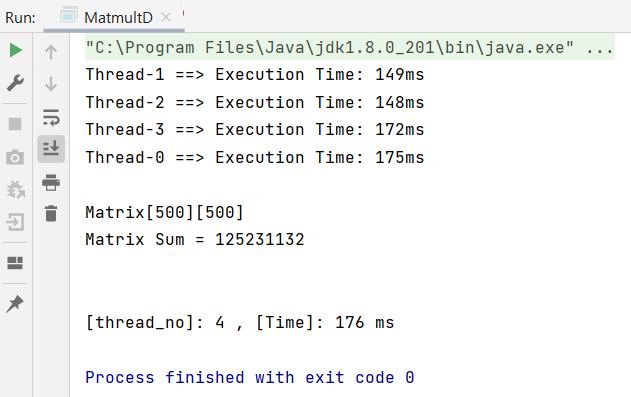
****

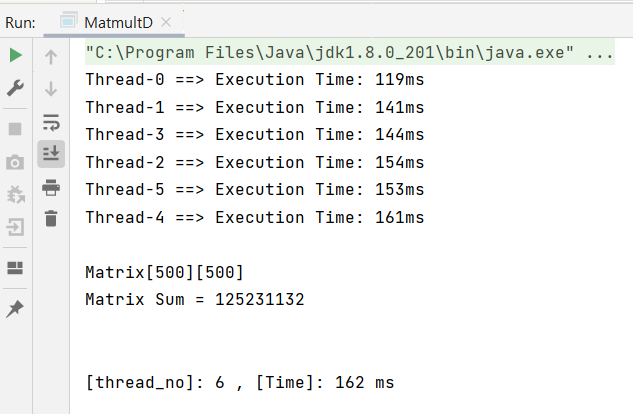
1. **Set the argument. In this picture, “4” means “thread\_no”. And pass the absolute file path of “mat500.txt” in the “Redirect Input from:”**
2. **Finally, run the source code.**

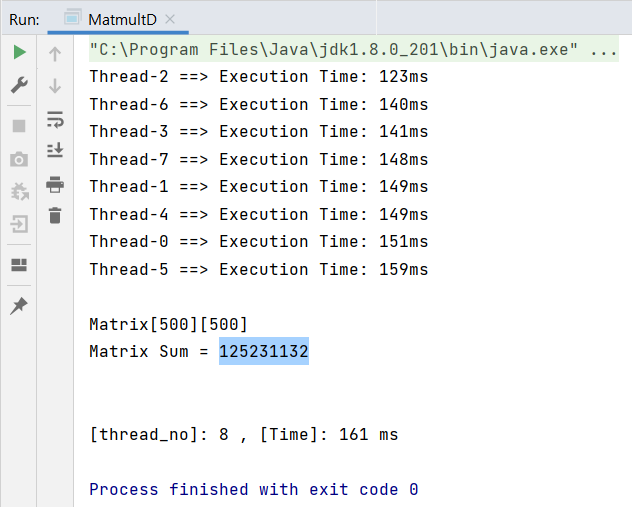
**[Results]**

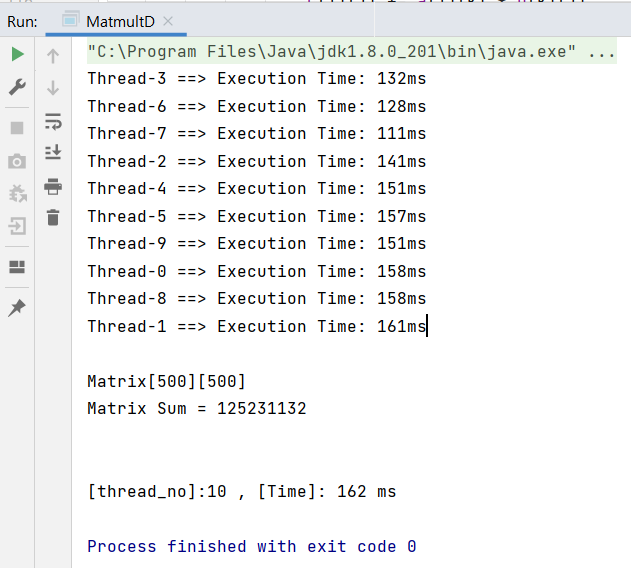


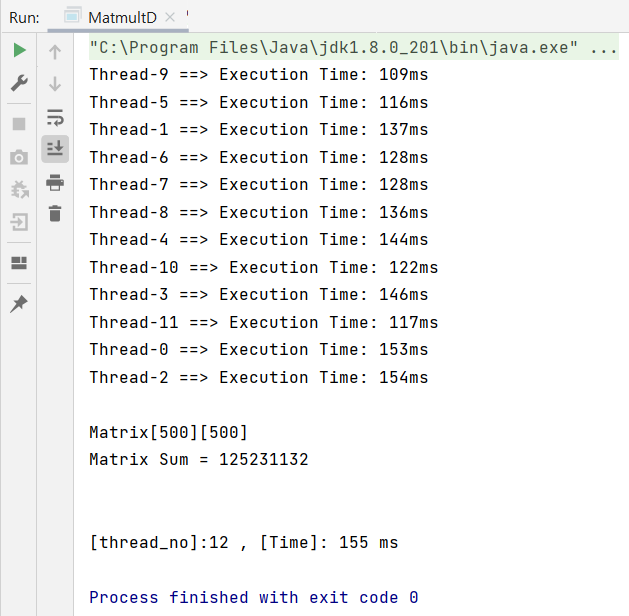


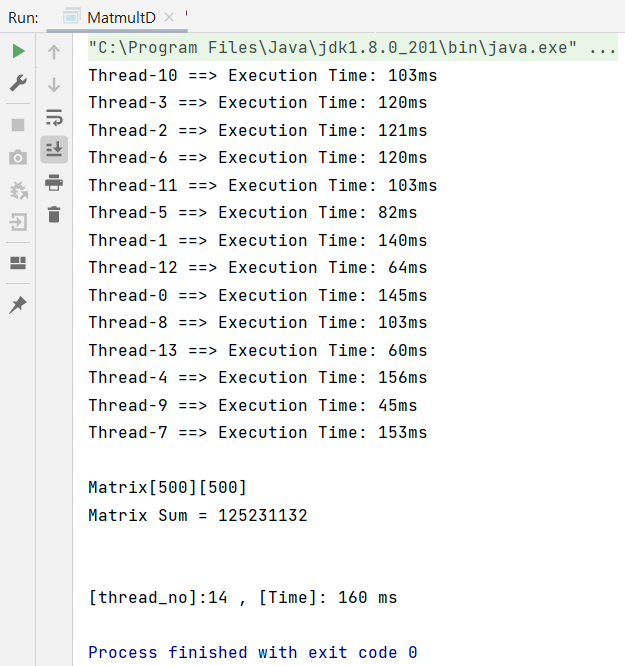


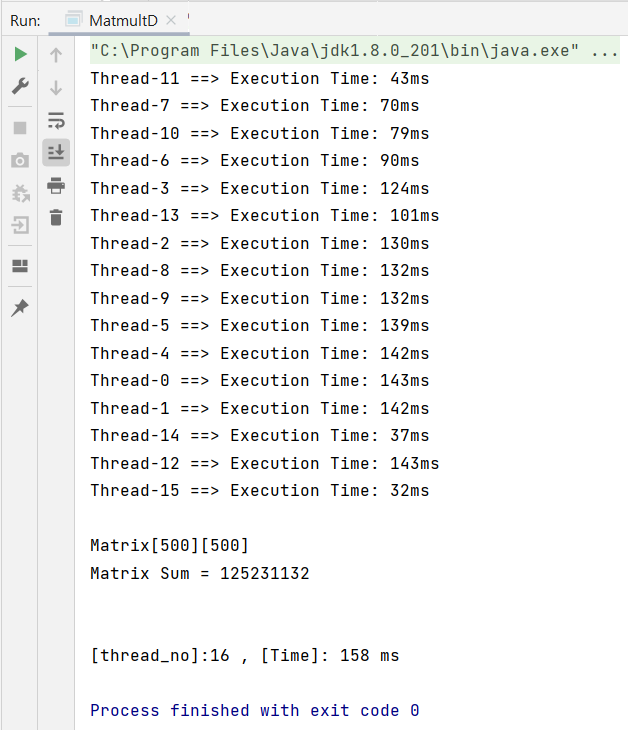


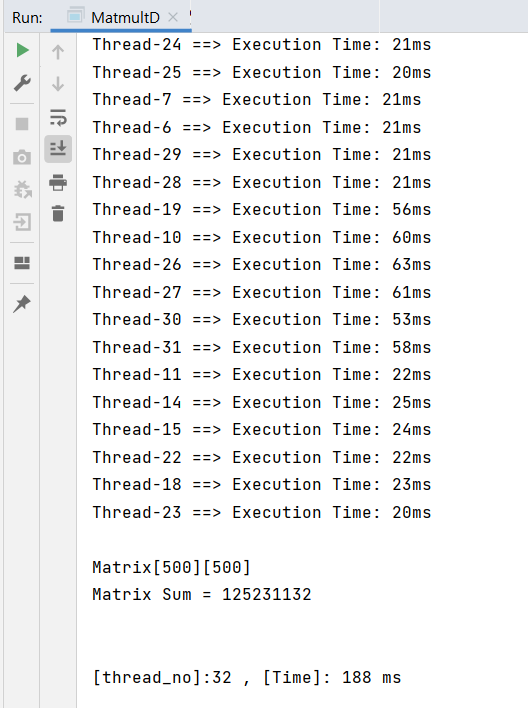












**Execution Time**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 32 |
| Exec time | 454ms | 268ms | 176ms | 162ms | 161ms | 162ms | 155ms | 160ms | 158ms | 188ms |

* **Performance**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 32 |
| Performance  (1/exec time) | 2.20e-3 | 3.37e-3 | 5.68e-3 | 6.17e-3 | 6.21e-3 | 6.17e-3 | 6.45e-3 | 6.25e-3 | 6.32e-3 | 5.31e-3 |

**[Explanation/Analysis on the Results]**

The above program uses cyclic decomposition method. Each thread deals with allocated rows. In the above program, when number of threads is small and increases (ex. 1, 2, 4, …), program performance is improved dramatically. But, when number of threads is large and increases (ex. 8, 10, 12, …), program performance is not improved. And number of threads becomes very large (ex. 32), performance eventually decreases. It’s because large number of threads makes large overhead, which makes program’s performance unable to increase. Also, very large number of threads is the same as sequential computation, not parallel computation. So, the performance becomes the same as execution time when number of threads is only one. For example, if number of threads is 320, then the execution time is similar to one which has only one thread. So, we need to decide how many threads we use.