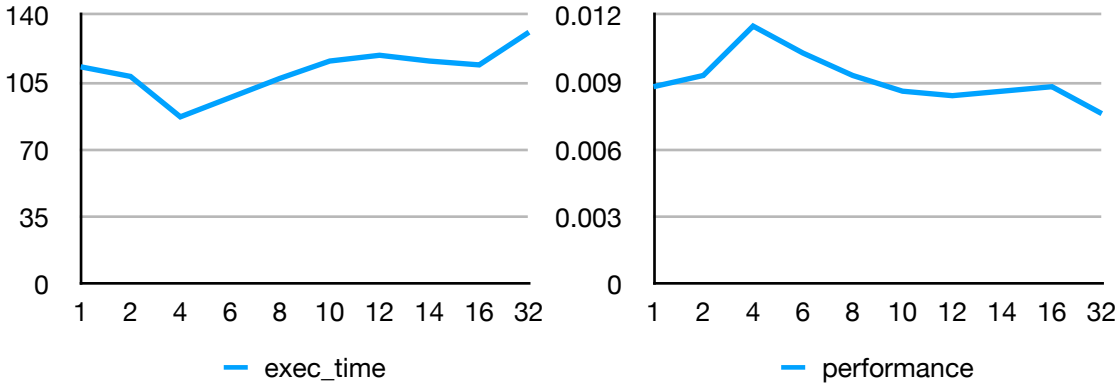
멀티코어컴퓨팅(Spring, 2023)
20171248 안재형

Multicore Project #1 - Problem02

Environment

Macbook Air(M1, 2020)
CPU : Apple M1 Chip(8-core CPU with 4 performance cores and 4 efficiency cores)
Memory : 8GB
OS : MacOS Ventura 13.2.1
IDE : IntelliJ

Execution Result



(unit : ms), task size of static(cyclic) and dynamic : 10

	1	2	4	6	8	10	12	14	16	32
exec_time	113	108	87	97	107	116	119	116	114	131

	1	2	4	6	8	10	12	14	16	32
perfor-mance	0.0088	0.0093	0.0115	0.0103	0.0093	0.0086	0.0084	0.0086	0.0088	0.0076

Analysis

1. CPU

I expected that performance would increase until 8 cores. However, the performance peaked when using 4 threads. There are some reasons that I can anticipate.

1. OS didn't assigned more CPU core to my program for some reason.
2. It needed much more resources than problem 1. So the efficiency cores may not be helpful for my program.
3. Bottleneck Problem - I'll explain in more detail in Analysis-No.4.

Also, I could see that as the number of threads exceeds the number of physical cores, performance has been decreased because of the increased overhead.

2. Variations of execution time

The execution time wasn't consistent between every execution.

One reason I can anticipate is that OS decided to execute my program earlier/later depending on it's situation. If there were some background tasks to do, my program would be executed slower than other times.

3. Calculation time of each threads

In the original code, while printing a thread's execution time, other threads could also print it simultaneously. Therefore, the result wasn't looking good. So I fixed it like the code below.

Original code

```
System.out.printf("thread_no: %d\n" , thread_no);
System.out.printf("Calculation Time: %d ms\n" , endTime-startTime);
```

Fixed code

```
System.out.printf("thread_no: %d\nCalculation Time: %d ms\n", thread, end-
Time-startTime);
```

4. Bottleneck Problem

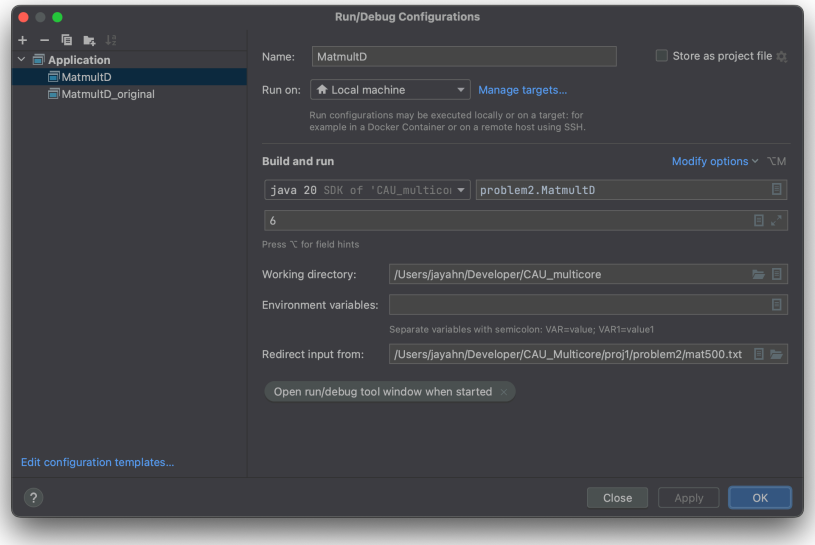
Because there were much more tasks than problem 1, as the number of threads is increasing, every thread should access more to synchronize. To do that, each thread should access more to MatmultD_Matrix_Result. However, it is protected with a lock. So there should be some bottleneck problem.

In my opinion, the main reason of performance decrease is due to this bottleneck problem.

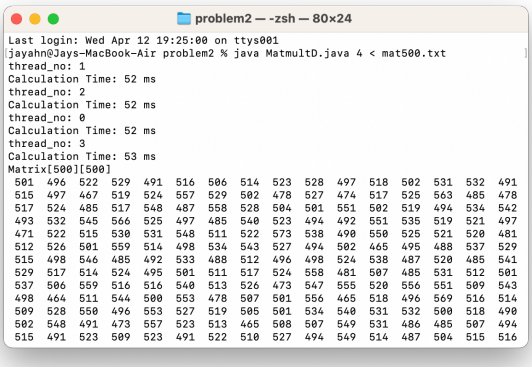
Execution

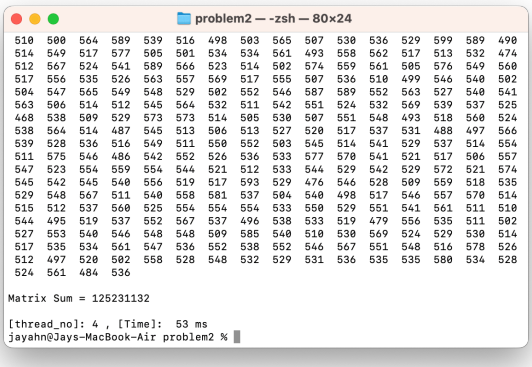
How to execute : java MatmultD.java 4 < mat500.txt

or, in Intellij, set Run/Debug configuration as the picture below.



Execution Results :





Code

MatmultD.java

```
public class MatmultD
{
    private static Scanner sc = new Scanner(System.in);
    public static void main(String [] args)
    {
        int thread_no=0;
        if (args.length==1) thread_no = Integer.valueOf(args[0]);
        else thread_no = 1;

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

        long startTime = System.currentTimeMillis();
        MatmultD_Matrix_Result result_matrix = new MatmultD_Matrix_Result(a.length,
b[0].length);

        // Thread creation
        MatmultD_Thread[] threads = new MatmultD_Thread[thread_no];
        for (int i=0; i<thread_no; i++) {
            threads[i] = new MatmultD_Thread(i, thread_no, a, b, result_matrix);
            threads[i].start();
        }
        for (int i=0; i<thread_no; i++) {
            try {
                threads[i].join();
            } catch (InterruptedException ignored) {
                System.out.println("Thread joining failed.");
            }
        }
        long endTime = System.currentTimeMillis();

        //printMatrix(a);
        //printMatrix(b);
        printMatrix(result_matrix.matrix);

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

```
    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("Matrix["+mat.length+"]["+mat[0].length+"]");
        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");
    }
}

class MatmultD_Matrix_Result {
    int[][] matrix;

    public MatmultD_Matrix_Result(int row, int column) {
        this.matrix = new int[row][column];
    }

    synchronized void set_result(int row, int column, int result) {
        matrix[row][column] = result;
    }
}

class MatmultD_Thread extends Thread {
    final int thread, num_of_threads;
    final int a[][], b[][];
    MatmultD_Matrix_Result result_matrix;
    public MatmultD_Thread(
            int thread,
            int num_of_threads,
            int a[][],
            int b[][],
            MatmultD_Matrix_Result result_matrix
    ) {
        this.thread = thread;
        this.num_of_threads = num_of_threads;
        this.a = a;
        this.b = b;
        this.result_matrix = result_matrix;
    }

    @Override
    public void run() {
        long startTime = System.currentTimeMillis();
        int n = a[0].length;
```

```
        int m = a.length;
        int p = b[0].length;

        for (int id_to_multiply = thread; id_to_multiply < m * p; id_to_multiply +=
num_of_threads) {
            int col = id_to_multiply % m;
            int row = (id_to_multiply - col) / m;

            int result = 0;
            for(int k=0; k<n; k++) {
                result += a[row][k] * b[k][col];
            }

            result_matrix.set_result(row, col, result);
        }
        long endTime = System.currentTimeMillis();
        System.out.printf("thread_no: %d\nCalculation Time: %d ms\n", thread, endTime-
startTime);
    }
}
```