Multicore Computing Project 1

Hardware and Software Information

Hardware Model	Lenovo ThinkPad T470s
Memory	7.5 GiB
Processor	Intel® Core™ i5-7300U CPU @ 2.60GHz × 4
Graphics	Mesa Intel® HD Graphics 620 (KBL GT2)
Disk Capacity	128.0 GB

OS Name	Fedora Linux 35 (Workstation Edition)
OS Type	64-bit
GNOME Version	41.5
Windowing System	X11
Software Updates	>

Hyperthreading: ON

```
Core Count: 2
Thread Count: 4
```

Problem 2

Tables

Execution Times

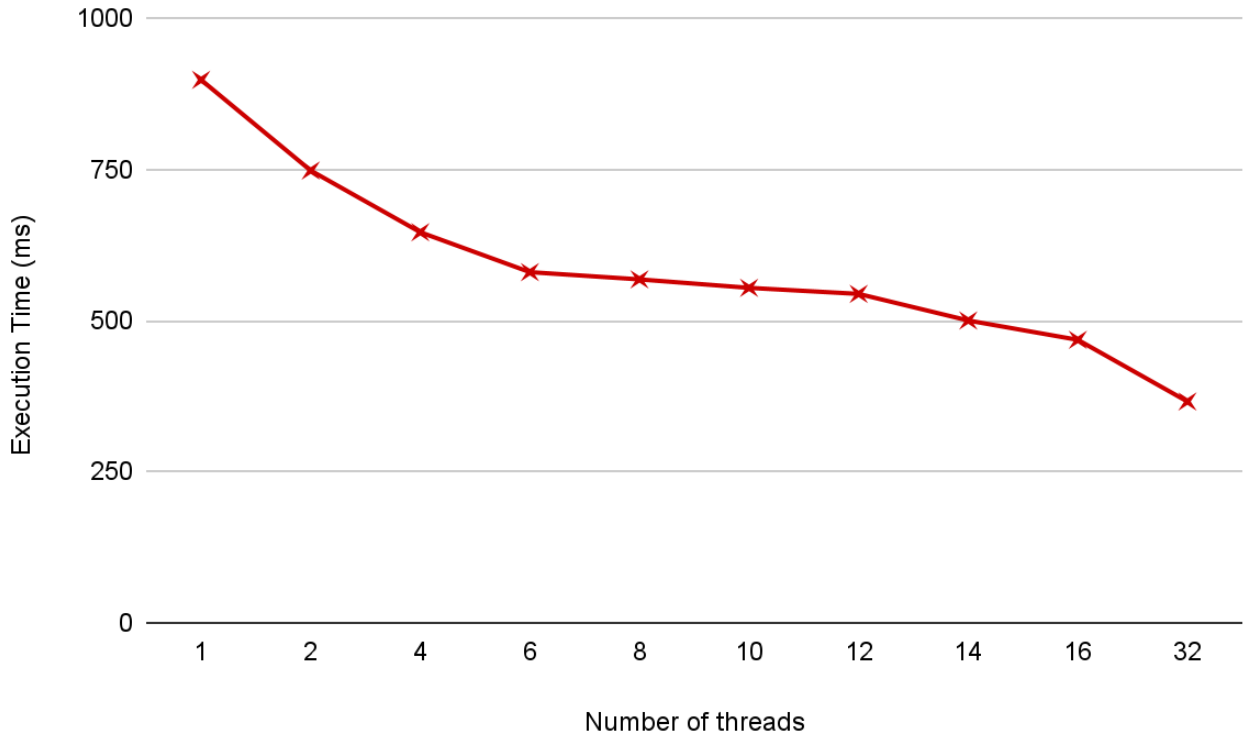
	1	2	4	6	8	10	12	14	16	32
exec times in ms	449	374	323	290	284	277	272	250	234	183

Performance

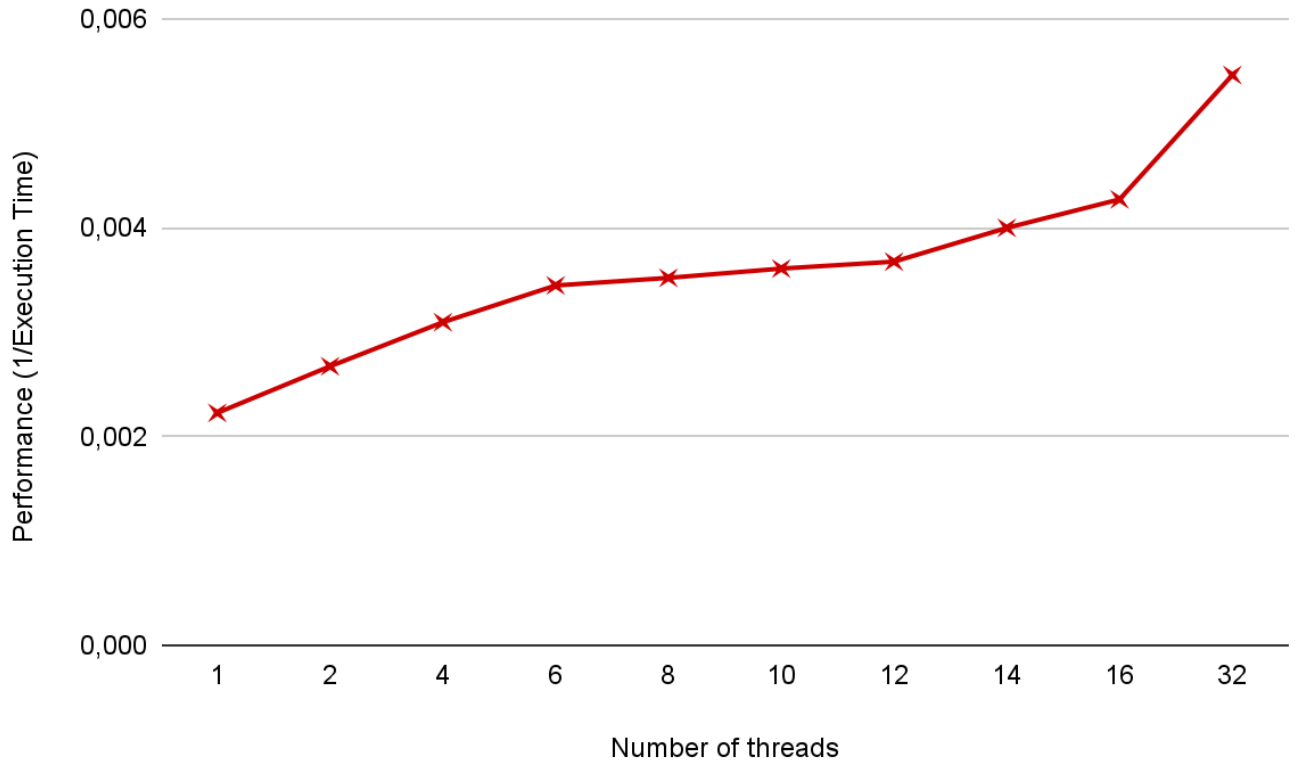
	1	2	4	6	8	10	12	14	16	32
Performance 1/exec time	0,00222 7171492	0,00267 3796791	0,00309 5975232	0,00344 8275862	0,00352 1126761	0,00361 0108303	0,00367 6470588	0,004	0,00427 3504274	0,00546 4480874

Graphs

Execution Times



Performance



Interpretation

As we can see on the graphs the more threads we add the faster the program gets. The program is definitely slower than the original one when it comes to low thread count (1 or 2) because there are many more variables to set up, however all this setup phase becomes worth it when we have a high thread count.

Compile & Run

Compile

To compile the code simply write:
javac [java_file]

Replace “[java_file]” by the name of the desired java file

Run

To run simply write:
java [class_name] [nbThreads] < [matrix_file]

Replace:

- [class_name] by the name of the compiled class
- [nbThreads] by the number of desired threads
- [matrix_file] by the file containing the matrices to compute

Code screenshots

```java
import java.util.*;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class MatmultD {
  private static Scanner sc = new Scanner(System.in);
  private static int nbThreads;
  private static int nbThreadsWorking;
  private static float tmpLinesPerThread;
  private static double linesPerThread;
  private static Matrix ans;
  private static Matrix matrixA;
  private static Matrix matrixB;

  public static void main(String[] args) {
    // Check args
    if (args.length == 1) {
      nbThreads = Integer.valueOf(args[0]);
    } else {
      nbThreads = 1;
    }

    // Creates matrices
    matrixA = new Matrix();
    matrixB = new Matrix();

    // Start program
    long startTime = System.currentTimeMillis();

    // Check if matrices are correct
    if (matrixA.height == 0) {
      long endTime = System.currentTimeMillis();
      System.out.printf("[nbThreads]:%2d , [Time]:%4d ms\n", nbThreads, endTime - startTime);
      return;
    }
    if (matrixA.width != matrixB.height)
      return;

    // Init final matrix
    ans = new Matrix(matrixB.width, matrixA.height);

    tmpLinesPerThread = matrixA.height / nbThreads;
    int rest = matrixA.height % nbThreads;

    if (tmpLinesPerThread < 1) {
      linesPerThread = 1;
      nbThreadsWorking = matrixA.height;
    } else if (rest != 0) {
      linesPerThread = Math.floor(tmpLinesPerThread);
      nbThreadsWorking = nbThreads;
    } else {
      linesPerThread = tmpLinesPerThread;
      nbThreadsWorking = nbThreads;
    }
    rest = matrixA.height % nbThreadsWorking;
    // Start Thread pool
    ExecutorService es = Executors.newCachedThreadPool();
    // Execute threads
    for (int i = 0; i < nbThreads; i++) {
      if (i < nbThreadsWorking) {
        if (i + 1 == nbThreadsWorking && rest != 0) {
          es.execute(new MyThread(i, (int) linesPerThread, rest));
        } else {
          es.execute(new MyThread(i, (int) linesPerThread));
        }
      } else {
        es.execute(new MyThread(i));
      }
    }

    // No more threads are expected to run
    es.shutdown();

    // Wait for all threads to finish
    while (!es.isTerminated()) {
    }

    // Stop chrono
    long endTime = System.currentTimeMillis();

    // Print
    printMatrix(ans.matrix);
    System.out.printf("[Number of threads]:%2d , [Process Time]:%4d ms\n", nbThreads, endTime -
startTime);
  }
```

```java
  public static int[][] readMatrix(int rows, int cols) {
    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");
  }

  public static class Matrix {
    public int height;
    public int width;
    public int[][] matrix;

    public Matrix() {
      height = sc.nextInt();
      width = sc.nextInt();
      matrix = readMatrix(height, width);
    }

    public Matrix(int width, int height) {
      this.height = height;
      this.width = width;
      this.matrix = new int[height][width];
    }

    public int getNumber(int y, int x) {
      return matrix[y][x];
    }
  }
```

```
  public static class MyThread implements Runnable {
    private int id;
    private int nbLines = 0;
    private boolean runnable = false;
    private int start = 0;

    public MyThread(int id, int lines, int more) {
      this.id = id;
      nbLines = lines + more;
      this.runnable = true;
      this.start = id * lines;
    }

    public MyThread(int id, int lines) {
      this.id = id;
      nbLines = lines;
      this.runnable = true;
      this.start = id * lines;
    }

    public MyThread(int id) {
      this.id = id;
      this.runnable = false;
    }

    public void run() {
      long startTime = System.currentTimeMillis();
      if (this.runnable) {
        int linesDone = 0;

        for (int i = start; linesDone < nbLines; i++) {
          for (int j = 0; j < matrixB.width; j++) {
            for (int x = 0; x < matrixB.width; x++) {
              ans.matrix[i][j] += matrixA.getNumber(i, x) * matrixB.getNumber(x,
j);          }
          }
          linesDone++;
        }
      }
      long endTime = System.currentTimeMillis();
      long timeDiff = endTime - startTime;
      System.out.println("Thread#" + id + " Execution Time: " + timeDiff + "ms");
    }
  }
}
```