

# 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

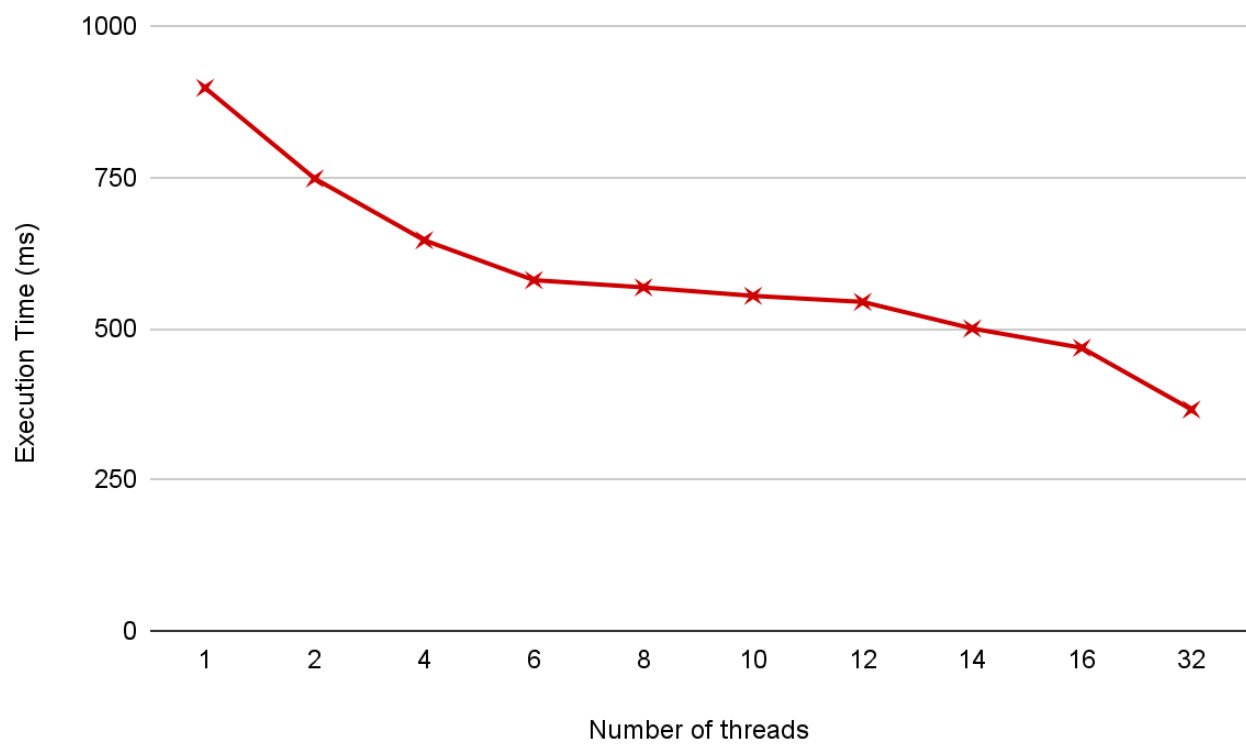
	1	2	4	6	8	10	12	14	16	32
exec times in ms	449	374	323	290	284	277	272	250	234	183

### Performance

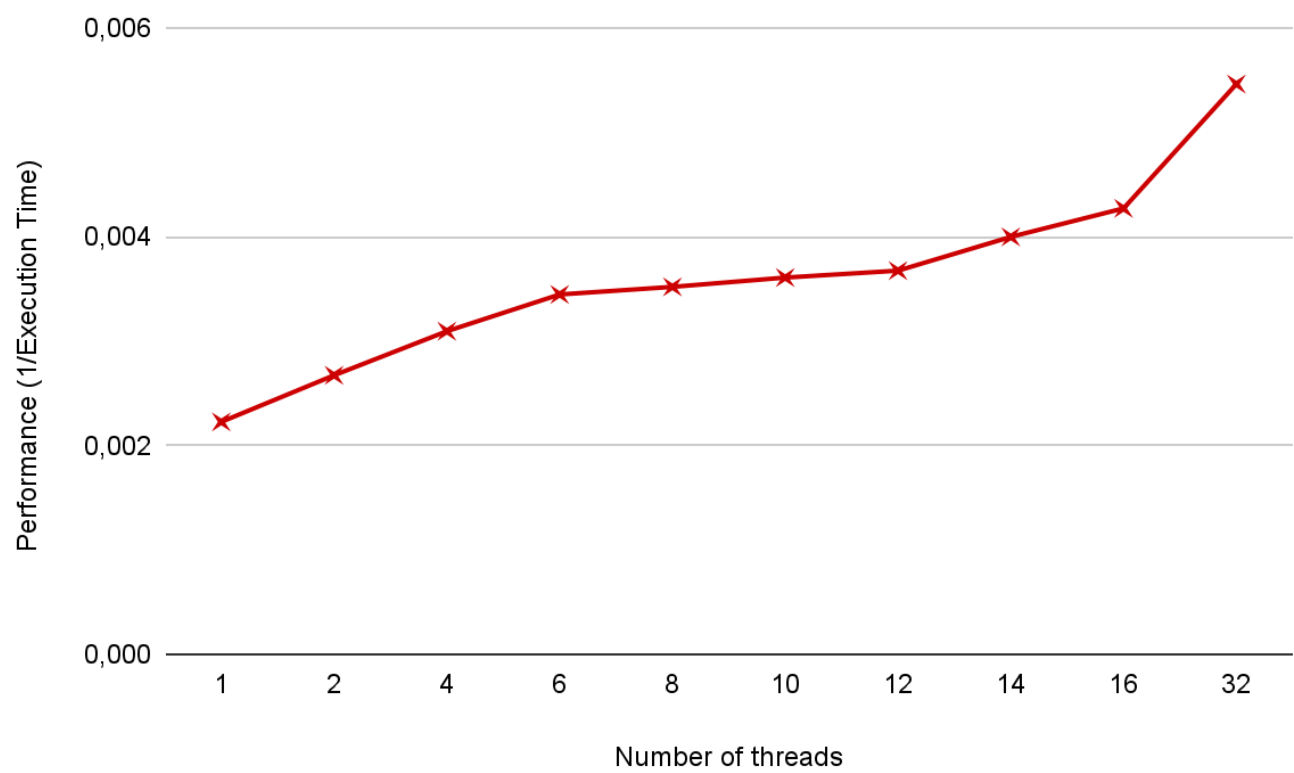
	1	2	4	6	8	10	12	14	16	32
Performance 1/exec time	0,002227171492	0,002673796791	0,003095975232	0,003448275862	0,003521126761	0,003610108303	0,003676470588	0,00400043504274	0,004273504274	0,005464480874

# 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

```
saber@fedora:~/multicore_computing/proj1/problem2

510 505 488 520 539 515 528 497 497 507 539 526 504 515 531 521 506 536 541 517 524 490 500 520 540 477 528 562 503 545 501 47
1 490 506 511 516 539 530 508 526 570 543 515 519 498 533 518 519 512 542 480 540 493 519 525 505 533 505 499 506 526 488 554 5
46 498 504 544 482 505 520 518 528 506 519 560 537 471 504 478 537 523 516 536 551 532 537 528 503 545 551 502 510 509 524 515
529 517 520 532 551 495 561 521 521 482 525 559 535 538 535 521 535 513 488 521 567 502 566 494 500 484 552 501 533 487 558 522
527 486 517 540 509 547 532 529 508 516 521 525 502 518 510 493 517 503 561 510 477 528 550 510 509 525 520 495 517 505 535 507
514 521 512 532 495 516 475 555 532 536 500 551 517 505 534 529 520 513 524 539 577 529 489 514 494 529 510 544 512 538 554 54
5 491 520 512 531 548 520 508 520 524 554 488 548 539 456 495 511 477 495 533 490 555 567 566 498 502 520 554 537 534 525 571 5
51 514 495 542 553 472 520 558 531 519 513 518 514 483 513 518 547 493 500 567 560 473 545 538 466 550 502 570 541 513 555 492
505 555 538 499 480 495 519 507 509 481 539 505 500 512 544 507 516 516 501 503 506 534 511 547 520 470 501 498 515 531 487 485
512 496 547 530 508 478 547 522 534 488 547 505 531 532 497 504 500 568 518 495 526 497 525 507 517 519 441 518 496 499 500 525
483 499 515 563 532 502 493 531 538 536 476 558 557 515 497 500 478 554 523 525 494 524 567 538 511 543 501 513 517 523 508 51
4 470 567 531 519 520 504 536 497 529 574 486 476 481 516 515 527 511 542 495 483 521 516 560 533 549 534 488 521 528 486 546 5
55 509 529 503 505 515 528 544 517 543 548 475 531 544 527 559 525 485 513 525 498 489 519 505 529 535 511 492 500 500 521 506
535 508 503 511 539 489 540 559 545 505 555 547 495 521 547 567 541 488 492 504 531 493 530 535 508 480 531 526 567 534 510 516
540 536 508 523 485 499 497 548 523 519 524 505 527 526 537 532 558 512 507 533 565 504 529
505 525 565 535 457 517 543 530 558 547 515 547 511 586 543 545 486 532 520 554 517 564 564 540 517 544 513 521 562 583 531 498
511 517 503 526 543 549 542 511 532 502 592 531 541 527 565 537 515 496 569 552 547 521 510 525 510 539 537 544 521 528 506 47
1 503 512 518 549 556 549 528 547 560 526 536 558 534 553 514 536 513 541 520 572 517 522 559 532 554 515 527 532 537 526 555 5
43 522 533 573 544 530 521 548 551 507 525 582 578 500 524 521 560 505 536 568 541 526 515 548 517 534 566 535 514 512 555 541
502 531 526 571 511 515 545 506 532 498 572 544 528 577 551 537 540 553 530 519 559 494 551 525 528 528 609 519 518 504 588 530
489 501 541 581 517 540 552 588 512 549 565 543 533 537 521 498 524 550 563 525 526 566 577 510 506 552 530 544 517 492 525 480
523 550 560 559 546 535 512 533 544 541 478 553 531 507 548 559 540 510 500 564 589 539 516 498 503 565 507 530 536 529 599 58
9 490 514 549 517 577 505 501 534 534 561 493 558 562 517 513 532 474 512 567 524 541 589 566 523 514 502 574 559 561 505 576 5
49 560 517 556 535 526 563 557 569 517 555 507 536 510 499 546 540 502 504 547 565 549 548 529 502 552 546 587 589 552 563 527
540 541 563 506 514 512 545 564 532 511 542 551 524 532 569 539 537 525 468 538 509 529 573 573 514 505 530 507 551 548 493 518
560 524 538 564 514 487 545 513 506 513 527 520 517 537 531 488 497 566 539 528 536 516 549 511 550 552 503 545 514 541 529 537
514 554 511 575 546 486 542 552 526 536 533 577 570 541 521 517 506 557 547 523 554 559 554 544 521 512 533 544 529 542 529 57
2 521 574 545 542 545 540 556 519 517 593 529 476 546 528 509 559 518 535 529 548 567 511 540 558 581 537 504 540 498 517 546 5
57 570 514 515 512 537 560 525 554 554 554 533 550 529 551 541 561 511 510 544 495 519 537 552 567 537 496 538 533 519 479 556
535 511 502 527 553 540 546 548 548 509 585 540 510 530 569 524 529 530 514 517 535 534 561 547 536 552 538 552 546 567 551 548
516 578 526 512 497 520 502 558 528 548 532 529 531 536 535 535 580 534 528 524 561 484 536

Matrix Sum = 125231132

[Number of threads]: 1 , [Process Time]: 526 ms
→ problem2 git:(main) |
```

## Code screenshots

```
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);
    }
}
```

```

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");
        }
    }
}
```