

ASSIGNMENT 2 Matrix multiplication using Block methods in Open MP and Java multithreading and multicore

Declaration

Name	Vikrant Singh
Student no.	21262315
Programme	MSc. In computing (secure software engineering)
Module Code	CA670
Assignment title	Assignment 2 - Java Threads vs. OpenMP - Matrix Multiplication
Submission Date	28/03/2022
Module Coordinator	David Sinclair

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. I have read and understood the Assignment Regulations set out in the module documentation. I have identified and included the source of all facts, ideas, opinions, and viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged, and the source cited are identified in the assignment references. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

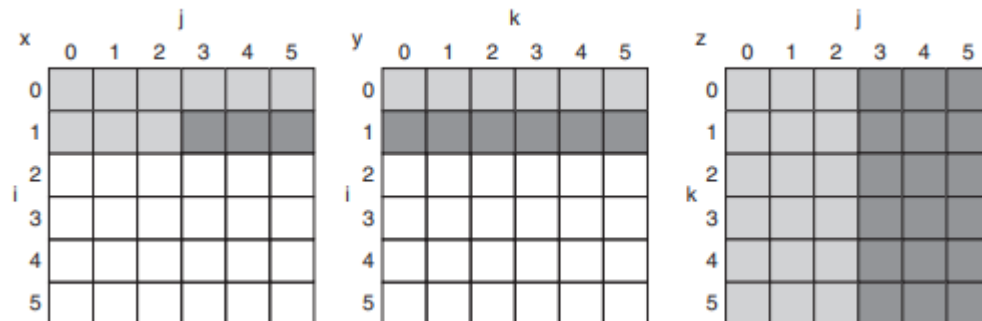
I have read and understood the referencing guidelines found recommended in the assignment guidelines.

Name: Vikrant Singh

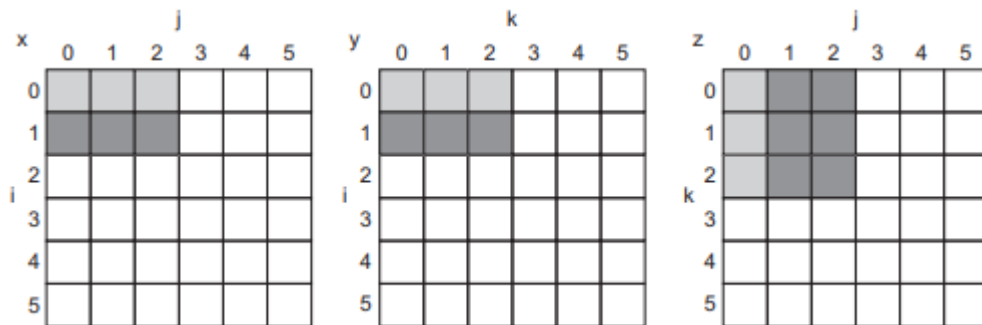
Date: 28/03/2022

Introduction

In both program the matrix multiplication method used is blocked method which is basically we divide large matrix into small submatrices, and it is a divided and conquer approach and our goal is to maximize accesses to the data loaded into the cache before data got replaced.



This figure [2] above shows the array access pattern with native approach where white colour blocks mean not yet accessed, light colour blocks mean older accesses, and dark colour blocks means newer accesses



This figure [2] above shows the accessed pattern of array with blocked method and we can see that a smaller number of elements are accessed.

Program Description

Open MP Program

1. Function 1

```
random_square_matrix(int dimension)
```

This function used to generate random square matrix for given dimension from varying from 200 to 2000 with increment of 200.

2. Function 2

```
zero_square_matrix (int dimension)
```

This function is used to create a matrix of same dimension as in above function, but this is empty matrix to store results.

3. Function 3

```
double ParallelWithPragma(NEW_TYPE** matrix_A, NEW_TYPE** matrix_B, NEW_TYPE**  
matrix_C, int size)
```

This function is like heart of the program and holds the #pragma functionality of OpenMP using following code

```
#pragma omp parallel shared(matrix_A, matrix_B, matrix_C, size, chunk) private(i, j, k, jj, kk, thread_no,  
tmp)
```

And it shares the variable specified in private for creating multiple instances of for loop with thread and each instance have its own private set of variables.

```
#pragma omp for schedule (static, chunk)
```

And the scheduling is used with static option with chunk size of 2 to schedule iteration of loop and the algorithm for matrix multiplication is there and then calculate total task completion time in seconds.

4. Function 4

```
void matrix_dimension(int dimension)
```

This function contains output functionality and call the above function to get the results and print them on console.

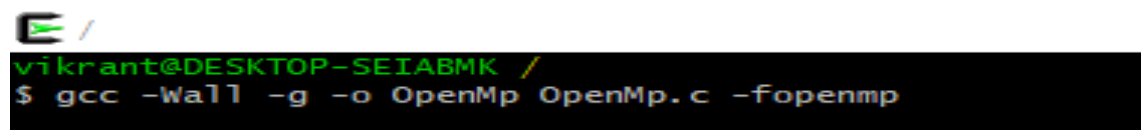
5. Function 5

```
int main(int argc, char* argv[])
```

Main contains loop which provide dimensions to *matrix_dimension(int dimension)* function to create different size matrices.

Output

For output I use my laptop which has i5 5th generation processor with only 4 cores and Cygwin which helps me un gcc on windows for open MP compilation.



```
vikrant@DESKTOP-SEIABMK /  
$ gcc -Wall -g -o OpenMp OpenMp.c -fopenmp
```

```

-----
Dimension = 200 x 200
-----
Number of pragma threads          |      4
Matrix Multiplication with pragma | 0.024597
-----
Dimension = 400 x 400
-----
Number of pragma threads          |      4
Matrix Multiplication with pragma | 0.178856
-----
Dimension = 600 x 600
-----
Number of pragma threads          |      4
Matrix Multiplication with pragma | 0.586845
-----
Dimension = 800 x 800
-----
Number of pragma threads          |      4
Matrix Multiplication with pragma | 1.450239
-----
Dimension = 1000 x 1000
-----
Number of pragma threads          |      4
Matrix Multiplication with pragma | 2.716245
-----
Dimension = 1200 x 1200
-----
Number of pragma threads          |      4
Matrix Multiplication with pragma | 4.741665
-----
Dimension = 1400 x 1400
-----
Number of pragma threads          |      4
Matrix Multiplication with pragma | 7.466387
-----
Dimension = 1600 x 1600
-----
Number of pragma threads          |      4
Matrix Multiplication with pragma | 11.683682
-----
Dimension = 1800 x 1800
-----
Number of pragma threads          |      4
Matrix Multiplication with pragma | 17.241550
-----
Dimension = 2000 x 2000
-----
Number of pragma threads          |      4
Matrix Multiplication with pragma | 24.064960
-----

```

Java Program with multithreading and multicore

1. Class 1

```
public class MatrixMultiplicationParallel{ }
```

```
ExecutorService executor=Executors.newFixedThreadPool(Runtime.getRuntime().availableProcessors());
```

```
executor.execute(new Block_matrix_multiply(matrixA, matrixB, result, dimension));
```

```
executor.shutdown();
```

This class contains multicore functionality using Executor as follows and initialize matrices required and create different dimension for different matrices with increment of 200 each with iteration.

2. Class 2

```
public class Matrix_generator{ }
```

This class creates matrices with different dimensions with random values and return the resulted matrix for multiplication.

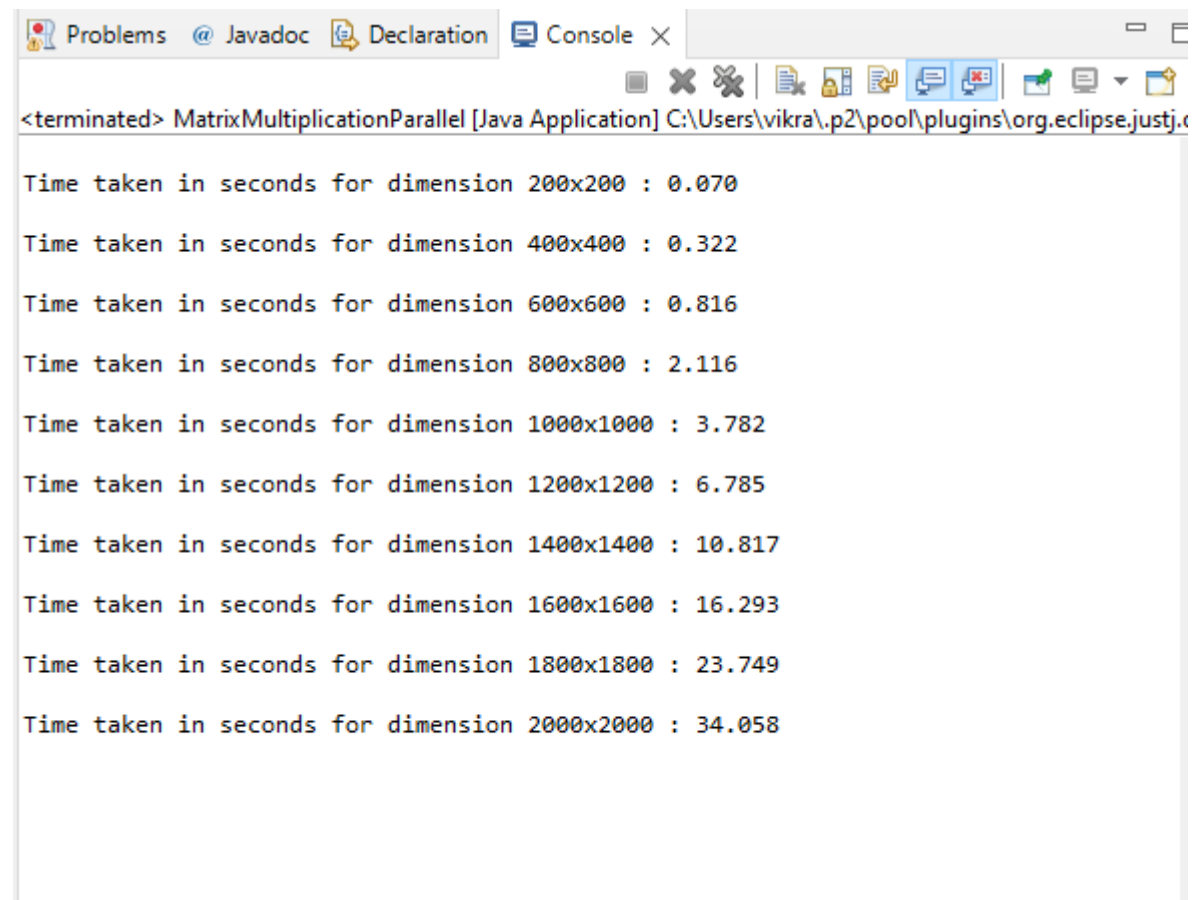
3. Class 3

```
public class Block_matrix_multiply implements Runnable{ }
```

This class is runnable so to make it multithreaded and contain blocked matrix multiplication algorithm which is exact same we are using with open MP.

Output

This program is made in Eclipse and run in Eclipse also using same laptop which has i5 5th generation processor with only 4 cores.



```
<terminated> MatrixMultiplicationParallel [Java Application] C:\Users\vikra\.p2\pool\plugins\org.eclipse.justj.c

Time taken in seconds for dimension 200x200 : 0.070
Time taken in seconds for dimension 400x400 : 0.322
Time taken in seconds for dimension 600x600 : 0.816
Time taken in seconds for dimension 800x800 : 2.116
Time taken in seconds for dimension 1000x1000 : 3.782
Time taken in seconds for dimension 1200x1200 : 6.785
Time taken in seconds for dimension 1400x1400 : 10.817
Time taken in seconds for dimension 1600x1600 : 16.293
Time taken in seconds for dimension 1800x1800 : 23.749
Time taken in seconds for dimension 2000x2000 : 34.058
```

Comparison of performance between two programs

Size of matrix	Results in java Multithreaded and multicore(seconds)	Results with pragma in Open MP (seconds)
200x200	0.070	0.023999
400x400	0.322	0.177599
600x600	0.816	0.570467

800x800	2.116	1.465169
1000x1000	3.782	2.736900
1200x1200	6.785	4.746159
1400x1400	10.817	7.507706
1600x1600	16.293	11.734908
1800x1800	23.749	17.387102
2000x2000	34.058	23.833347

Reference

- [1] A. Kalapgar, "Github," 19 April 2020. [Online]. Available: https://github.com/iarchanaa/Large_Matrix_Multiplication_Using_openMP. [Accessed 03 2022].
- [2] J. L. H. a. D. A. Patterson, "chapter 2 Blocking," in *Computer Architecture A Quantitative Approach (5th edition)*.
- [3] J. Corner, "OpenMP: Data-Sharing Rules," 27 June 2016. [Online]. Available: [http://jakascorner.com/blog/2016/06/omp-data-sharing-attributes.html#:~:text=%23pragma%20omp%20parallel%20for%20shared\(n%2C%20a\)%20private,p%20variables%20is%20sometimes%20unintuitive..](http://jakascorner.com/blog/2016/06/omp-data-sharing-attributes.html#:~:text=%23pragma%20omp%20parallel%20for%20shared(n%2C%20a)%20private,p%20variables%20is%20sometimes%20unintuitive..) [Accessed 03 2022].
- [4] J. Corner, "OpenMP: For & Scheduling," 13 June 2016. [Online]. Available: <http://jakascorner.com/blog/2016/06/omp-for-scheduling.html>. [Accessed 03 2022].
- [5] Venkatesh, "Matrix Multiplication with Java Threads - Optimized Code (Parallel)," 22 January 2020. [Online]. Available: <https://www.javaprogramto.com/2020/01/java-matrix-multiplication-threads.html>. [Accessed 03 2022].