Large Dense Matrix Multiplication
Speeding Up Computation Using Concurrency

Hadrien Bailly
Dublin City University
M.Sc. in Computing
hadrien.bailly2@mail.dcu.ie (21266176)

An assignment submitted to Dublin City University, School of Computing for module CA670 Concurrent Programming.

I understand that the University regards breaches of academic integrity and plagiarism as grave and serious. I have read and understood the DCU Academic Integrity and Plagiarism Policy. I accept the penalties that may be imposed should I engage in practice or practices that breach this policy. I have identified and included the source of all facts, ideas, opinions, viewpoints of others in the assignment references. Direct quotations, paraphrasing, discussion of ideas from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the sources cited are identified in the assignment references.

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. By signing this form or by submitting this material online I confirm that 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. By signing this form or by submitting material for assessment online I confirm that I have read and understood DCU Academic Integrity and Plagiarism Policy.

Name: Hadrien BAILLY
Date: March 16, 2022

CONTENTS

ASSIGNMENT

Java Threads vs. OpenMP - Matrix Multiplication

Description

You are to develop 2 programs, capable of executing on multiple cores, that can multiply 2 large dense matrices. The 1st program is a multithreaded Java program. The 2nd program is an OpenMP program. Both programs should be efficient and not adopt a naive approach.

Submission

You should submit the following files by email by 10am on Monday 4th April 2022.
The source code for both programs in separate files. A Word file that includes:

- A declaration that the submitted work (Java programs and Word file) are solely the work of the student except for elements that are clearly identified, cited and attributed to other sources.
- A description of the design of both algorithms.
- The results of testing both algorithms with various sizes of matrices and a variable number of cores.

Marks

Marks will be awarded for:

- The selection and description of the chosen algorithms.
- The implementation of these algorithms.
- The tests performed on the algorithms.

The assignment carries 15 marks and late submissions will incur a 1.5 mark penalty for each 24 hours after the submission deadline.

All submissions will be checked for plagiarism and severe penalties will apply.

I. Introduction

In this document, we will present you with two programs of matrix multiplication, in Java and C language. Each program will feature a sequential and parallel implementation, which we will compare in terms of design and performance.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 & 64 \\ 139 & 153 \end{bmatrix}$$

Figure 1: Rectangular Matrix Multiplication: Example

In the first section of this document, we will start by analysing and specifying the requirements of matrix multiplication. That is, we will investigate what are the tasks to realize and to what extent they can be parallelized. Once the tasks are clearly defined, we will present three different algorithms to solve the problem: using a plain sequential approach, and two parallelized approaches with distinct concurrency levels.

In the second section, we will move on to our first language, Java, and program. We will present its design and the implementation of the three algorithms. In addition, we will discuss a fourth implementation using `ReentrantLock`, observe that it fails at handling dense matrices, and offer some keys to explain why it is underachieving.

In the third section, we will turn to the next programming Language, C. For the realization of the C program, we had to build a number of dedicated libraries, which we will present and review quickly. We will then discuss again the core design and implementation of the matrix multiplication algorithms.

In the final section, we will review the performance of both programs: we will run both program against a number of matrices of increasing density, and record the time of execution. We will then use the recorded durations to ”benchmark” implementations against each other:

- Between parallel and sequential algorithms ;
- Between programming languages ;

and discuss our findings.

> We expect the parallel implementations to achieve a higher throughput than the sequential ones.

Finally, we will comment on our overall work on parallel matrix multiplication and offer some elements of critique.

II. Analysis

"*In mathematics, a matrix (plural matrices) is a rectangular array or table of numbers, symbols, or expressions, arranged in rows and columns, which is used to represent a mathematical object or a property of such an object.*" (Wikipedia 2022)

A. Definition

A matrix is

- A rectangular (*bi-dimensional*) array
- composed of values (*numbers*)
- arranged in (*identified by*) rows and columns.

Matrix multiplication involves two matrices and computes the *dot product* of the values from two matrices. It is only defined when the left operand is an m-by-n matrix and the right operand an n-by-p matrix (Wikipedia 2022). It is also non-commutable.

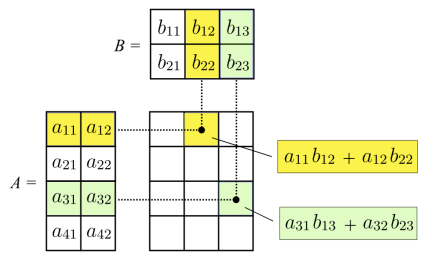

Figure 2: Matrix Multiplication by Dot Product (Svjo 2022)

Let it be the following matrix multiplications $A \times B$, coloured by vectors:

Figure 3: Matrix with colour identified vectors

The size of the vectors used in the dot product is equal to the width of the first matrix and height of the second. In this case, it is of size 3.

In order to compute the product of these matrices, the following operations are needed:

- $\langle 1, 2, 3 \rangle \times \langle 7, 9, 11 \rangle$
- $\langle 4, 5, 6 \rangle \times \langle 7, 9, 11 \rangle$
- $\langle 1, 2, 3 \rangle \times \langle 8, 10, 12 \rangle$
- $\langle 4, 5, 6 \rangle \times \langle 8, 10, 12 \rangle$

Or, in other words, a sum of products:

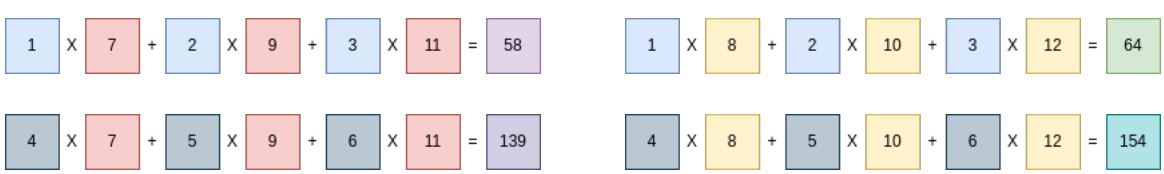

Figure 4: Matrix Multiplication Operations

> In the following pages, we will refer to each combination of vectors from a matrix as a *stripe*.

B. Algorithms

The problem with matrix multiplication, and the dot product, is the accumulation of the products: when using binary operations, products can be calculated instantly and independently. Sums on the other hand require a number of consequent operations.

$$\begin{aligned} a_0 b_0 &= (\mathbf{1} \times \mathbf{7}) + (\mathbf{2} \times \mathbf{9}) + (\mathbf{3} \times \mathbf{11}) \\ &= (\mathbf{7} + \mathbf{18}) + 33 \\ &= (\mathbf{25} + \mathbf{33}) \\ &= \boxed{\mathbf{58}} \end{aligned}$$

1) Sequential: In order to compute the dot product, the simplest approach is the sequentialization of all stripes computations.

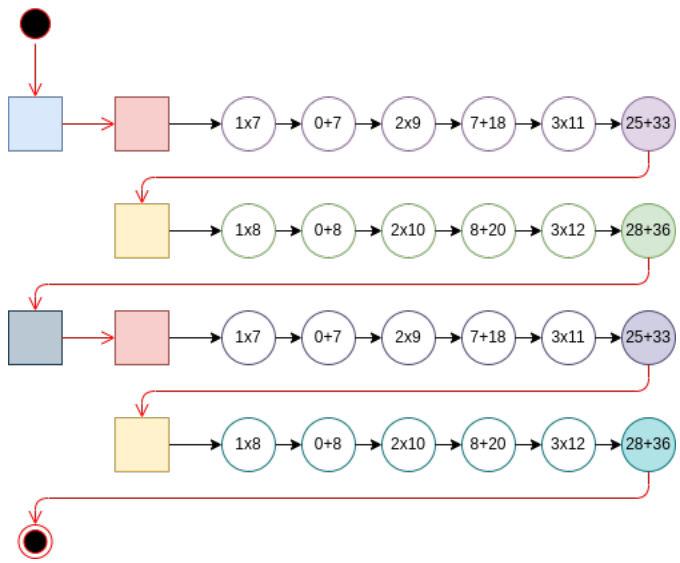

Figure 5: Sequential Multiplication Algorithm

The algorithm proceeds sequentially by processing all operations from a given stripe, before moving to the next column (or row).

Complexity = $|rows_A| \times |width_B| \times (2 \times |vector|) \quad \Leftrightarrow \quad m \times 2n \times p$ operations per thread (baseline).

> This method does not take advantage of the fact that all multiplication operations can occur concurrently and the addition operations in any order.

2) Parallel Stripes: In order to speed up the computation of the matrix multiplication, the first step is to identify the parallelizable tasks: the contention is located at the intersection of each row and column, where the addition operations take place.

In our example matrix multiplication, there are four contention points.

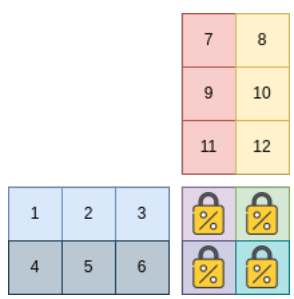

Figure 6: Matrix Contention Points

When a task is currently writing to one of the cell of the product matrix, nothing prevents another task to concurrently accumulate and write into another cell of the same matrix.

A parallel algorithm could thus take the following form:

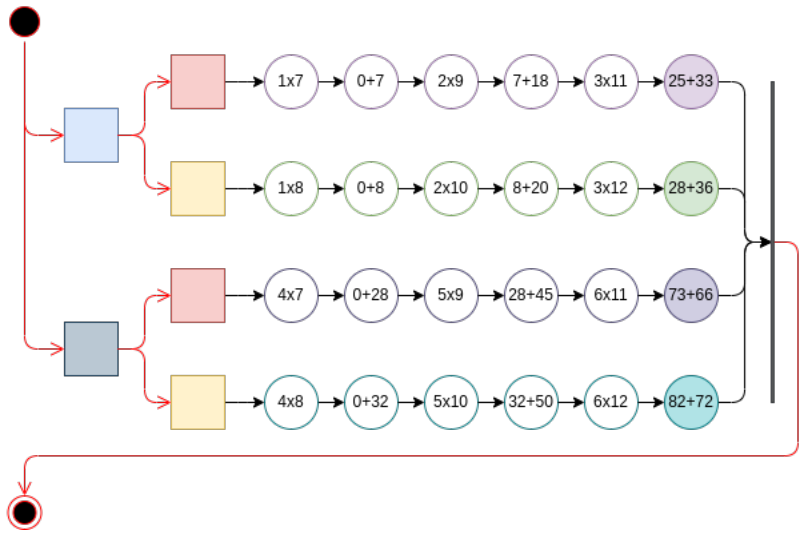

Figure 7: Parallel Multiplication Algorithm With Sequenced Tasks

Each stripe is processed as a parallel task on a different thread: for each stripe, all multiplication and addition operations are processed sequentially, and all stripes are executed concurrently. The process is completed when all threads have joined.

Complexity $= \quad 2 \times |vector| \quad \Leftrightarrow \quad 2n$ operations per thread.

3) Parallel Reduction: Parallelism can even be increased further up by decoupling multiplication operations from the addition operations: the former can be computed independently, and the latter require to be processed incrementally (*flow dependency*).

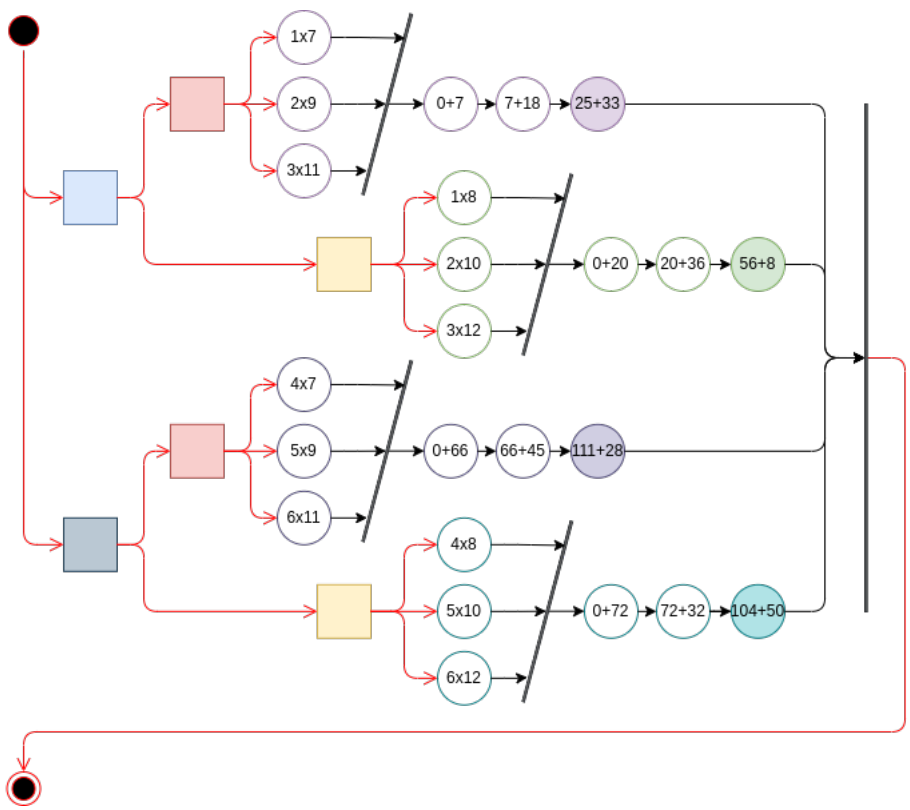

Figure 8: Matrix Multiplication Algorithm With Inordered Reduction Operations

For each stripe, all multiplications are executed on a different thread, then each thread compete for writing in inorder to the contention point (critical region).

Complexity $= 2$ operations (multiply-accumulate) per thread.

C. Comment

According to our analysis, three distinct algorithms can be followed to resolve a matrix multiplication.

- 1 plain sequential algorithm.
- 2 parallel algorithms.

The main leverage point for parallelizing the computation of a matrix multiplication is the independence of the stripes (pairs of vectors used in the calculation of a single cell of the product matrix): it can be processed parallelly without requiring the set-up of contention control on a critical region.

- Since the input matrices are not modified during the computation of the product, each row and column of the input matrices can be read simultaneously by any number of threads.
- Since each thread is responsible for one stripe, it has exclusive access to the target cell in the product matrix, and therefore is not at risk for racing.

Taken to its more extreme version, parallelism can even be brought up to the individual treatment of each multiplication in each stripe, at the expense of managing the competition when accumulating the result to the product matrix. This notably leads to an explosion of the number of threads.

> Here we did not take into account the overhead incurred by the generation of new threads nor the number of cores to estimate the performance of the algorithms. This will be discussed in the last section of this document instead.

In the next two sections, we will present how we have implemented these two algorithms in Java and C programming languages.

III. Java

We will now present you with our Java implementation of the matrix multiplications algorithms.

We will first introduce our overall application design and architecture, and present the `Matrix` and `MatrixLoader` classes, and `MatrixMultiplication` interface. Then, we dive into each of the concrete implementation of the `MatrixMultiplication` interface and detail the steps involved in the computation. Finally, we will describe the tests that we have implemented to validate and stress test our implementations, and show the measures collected.

A. General Design

Our Java program is composed of nine classes distributed across three packages, plus one test class.

- **`algorithms`** contains all the classes pertaining to the implementation of the matrix multiplication.
  - `Algorithm`
    An enumeration presenting the different implementations of the matrix multiplication.
  - `MatrixMultiplication`
    An interface describing the contract for the implementation of the matrix multiplication.
  - `SequentialMatrixMultiplication`
    A class implementing the simple, plain sequential computation.
  - `ParallelStripeMatrixMultiplication`
    A class implementing the first parallel algorithm, under which stripes are processed concurrently.
  - `ParallelReduceMatrixMultiplication`
    A class implementing the second parallel algorithm, under which all multiplication operations are realized concurrently.
  - `ParallelStreamMatrixMultiplication`
    A class implementing the first parallel algorithm using the *Java Stream* interface.
- **`matrices`** containing all the classes pertaining to the representation of matrices in the JVM heap.
  - `Matrix`
    A class representing a bi-dimensional matrix composed of integers.
  - `MatrixLoader`
    A matrix utility class used to load given matrices from a directory into memory.
- **`utils`** containing classes with an accessory function to the main feature of the program.
  - `Chronometer`
    A utility class for estimating the duration between two events in microseconds.
- **`tests`** the two tests classes.
  - `MultiplicationTestByAlgorithm`
    A parametrized test class for testing each algorithm against a number of matrices.
  - `MultiplicationTestByMatrix`
    A parametrized test class for testing each matrix group against all algorithms.

1) *Matrix Class:*

Listing 1: Matrix.java

```java
/**
 * A class representing a bi-dimensional matrix composed of
     integers.
 *
 * @author Hadrien BAILLY.
 */
@Slf4j
@Data
public class Matrix {

    /**
     * The name of the matrix
     */
    private String name;
    /**
     * The inner, raw representation of the matrix.
     */
    private int[][] values;
    /**
     * The number of rows in the matrix.
     */
    private int height;
    /**
     * The number of columns in the matrix.
     */
    private int width;

    /**
     * A constructor with some logging.
     * @param name the name of the matrix to create.
     */
    public Matrix(final String name) { ... }

    /**
     * Allocate the given memory space to the current
         matrix.
     * @param values the memory space to allocate.
     */
    public void allocate(final int[][] values) { ... }

    /**
     * Checks if the given input is valid for representing
         a matrix.
     * @param values the inner representation of a matrix.
     * @return true if the values are a valid
         bi-dimensional matrix, false otherwise.
     */
    private boolean isValidInput(final int[][] values) {
        ... }

    /**
     * Fills the memory space of the matrix with the given
         values.
     * @param rows the values to use for filling the memory.
     */
    public void fill(final List<String[]> rows) { ... }

    /**
     * Check if the current matrix is equivalent to the
         given matrix.
     * @param that the given matrix to which compare
         against equivalence.
     * @return true if the two matrices have the same
         dimensions and contents, false otherwise.
     */
    public boolean matches(Matrix that) { ... }

    /**
     * Display the current matrix on screen.
     * CAUTION: on large matrices, may cause the machine to
         become unresponsive.
     */
    public void print() { ... }

    /**
     * Return the value at the given coordinates in the
         current matrix.
     * @param row the row ordinate x.
     * @param column the column ordinate y.
     * @return the value contained at [x,y].
     * @throws IllegalArgumentException when the
         coordinates are invalid (not contained in the
         matrix).
     * @see #contains(int, int)
     */
    public int get(final int row, final int column) { ... }

    /**
     * Check if the given coordinates are contained in the
         current matrix.
     * @param row the row ordinate x.
     * @param column the column ordinate y.
     * @return true if the coordinates [x,y] are contained
         in the table, false otherwise.
     */
    public boolean contains(final int row, final int
        column) { ... }

}
```

The matrix class contains all the code need to represent and manipulate the integer values of a matrix in a JVM environment.

Our current implementation is limited to integer representation. If an input matrix contains large integers or a very high number of values in a vector, this may lead to integral overflow and incorrect values. It could have been possible to use long integers instead, but this would be at the expense of memory (increase by a factor two) and weight heavier on the heap. This is the reason why we choose for our current implementation to restrict to simple integers.

In addition, we chose to use a bi-dimensional array to represent the data of the matrix. This is expected to reduce the overhead of lookups (which would occur with a list of lists) and technically allow the data to be contiguous in memory. We know that this is not always true with memory paging, but on reasonably small matrices this can still play in favour of speed.

2) MatrixMultiplication Interface:

Listing 2: MatrixMultiplication.java

```java
/**
 * An interface describing the contract for the
     implementation of the matrix multiplication.
 *
 * @author Hadrien BAILLY
 */
public interface MatrixMultiplication extends
     BiFunction<Matrix, Matrix, Matrix> {

    /**
     * Compute the product of two given matrices.
     * @param a the left operand matrix.
     * @param b the right operand matrix.
     * @return a new [x,y] matrix, where x is the height of
         matrix A, and y the width of matrix B, containing
         the result of the dot product multiplications of
         all vectors in A and B.
     * @throws IllegalArgumentException if the operands
         matrices dimensions do not match, i.e. the width
         of A is not equal to the height of B.
     * @see #canMultiply(Matrix, Matrix)
     */
    default Matrix compute(final Matrix a, final Matrix b) {
        if (canMultiply(a, b)) {
            return apply(a, b);
        }
        throw new IllegalArgumentException("Cannot multiply
             matrices");
    }

    /**
     * Check if the dimensions of two given matrices
         matches for multiplication.
     * @param a the left operand matrix.
     * @param b the right operand matrix.
     * @return true if the width of A is equal to the
         height of B.
     */
    default boolean canMultiply(final Matrix a, final
         Matrix b) {
        return a.getWidth() == b.getHeight();
    }

}
```

The matrix interface is a simple artifice to facilitate the use of the different implementation in our parametrized tests: instead of manually testing each implementation individually, we wrote our tests against the interface, then execute the tests by iterating over the list of classes implementing the interface. This is used in combination with the `Algorithm` enumeration, which effectively lists these classes.

3) Algorithm Enumeration:

Listing 3: Algorithm.java

```java
/**
 * An enumeration presenting the different implementations
     of the matrix multiplication algorithms.
 *
 * @author Hadrien BAILLY
 */
@RequiredArgsConstructor
public enum Algorithm {
    /**
     * The sequential algorithm.
     */
    SEQUENTIAL("Sequential", new
         SequentialMatrixMultiplication()),
    /**
     * The simple parallel algorithm computing stripes
         concurrently.
     */
    STRIPE("Parallel Striping", new
         ParallelStripeMatrixMultiplication()),
    /**
     * The deeper parallel algorithm computing
         multiplication concurrently and addition
         consequently.
     */
    REDUCE("Parallel Reduction", new
         ParallelReduceMatrixMultiplication()),
    /**
     * A revised version of the simple parallel algorithm
         computing stripes concurrently, using Java Streams.
     */
    STREAM("Java Streams parallelism", new
         ParallelStreamMatrixMultiplication());

    /**
     * The short name for the multiplication algorithm.
     */
    private final String name;
    /**
     * An instance of the multiplication algorithm.
     */
    public final MatrixMultiplication implementation;
    /**
     *
     * @return the name of the algorithm
     */
    public String getName() {
        return name;
    }

    /**
     * Execute a matrix multiplication following the
         current algorithm implementation.
     * @param a the left operand matrix.
     * @param b the right operand matrix.
     * @return a product matrix.
     */
    public Matrix multiply(final Matrix a, final Matrix b) {
        Logging.log.info("Computing product of {} and
             {}...", a.getName(), b.getName());
        return implementation.compute(a, b);
    }
}

/**
 * A utility class holding a private logger for the
     interface.
 */
@NoArgsConstructor(access = AccessLevel.PRIVATE)
final class Logging {

    /**
     * The private logger for the interface.
     */
    static final Logger log =
         LoggerFactory.getLogger(Algorithm.class);

}
```

This enumeration is used to list the implementation of the matrix multiplication algorithms and to associate them with a single instance of the related class. Again, this is used to perform multiplication tests automatically.

B. Implementation

As presented above, we have split our implementation of the algorithm in four classes:

1) `SequentialMatrixMultiplication`
2) `ParallelStripeMatrixMultiplication`
3) `ParallelReduceMatrixMultiplication`
4) `ParallelStreamMatrixMultiplication`

Here, we will not describe these classes in full length, but instead provide insight on the implementation choices made.

1) Sequential: The sequential algorithm implementation is very straightforward.

We simply imbricated a number of `for` loops and left the main thread iterate over each of the triples $\langle row, colum, stripe \rangle$. This is very similar to that of the C implementation.

Listing 4: SequentialMatrixMultiplication#*execute*

```
final int[][] values = product.getValues();
for (int column = 0; column < widthB; column++) {
    for (int row = 0; row < heightA; row++) {
        values[row][column] = computeCell(a, b, row, column);
    }
}
```

2) Stripes: The implementation of the algorithm based on the stripe parallelization is a little more advanced, and involves two private subclasses.

- `Context`, the main class that organizes the parallel execution and memorizes the shared information.
- `Task`, the class that represents the work to be executed by a single tread in a given context, and contains the private information on the current triple $\langle row, colum, stripe \rangle$ being executed.

A main method controls the execution of the algorithm...

Listing 5: ParallelStripeMatrixMultiplication#*execute*

```
public Matrix execute() {
    log.info("Executing multiplication...");

    chronometer.start();

    generateAndExecuteTasks();
    awaitCompletion();

    chronometer.stop();
    log.info("All tasks completed.");

    log.info("Duration: {} ms", String.format("%,d", chronometer.getDuration()));

    return result;
}
```

... and is assisted by two methods: one that organizes and launches each of the tasks needed of the overall computation, and another that ensures that all tasks have completed before returning the result.

Listing 6: ParallelStripeMatrixMultiplication#*generateAndExecuteTasks*

```
private void generateAndExecuteTasks() {
    log.info("Generating and executing tasks...");
    chronometer.start();
    for (int row = 0; row < height; row++) {
        for (int column = 0; column < width; column++) {
            final Task task = new Task(this, row, column);
            service.execute(task);
        }
    }
    log.info("Tasks generated.");
}
```

Listing 7: ParallelStripeMatrixMultiplication#*awaitCompletion*

```
private void awaitCompletion() {
    log.info("Awaiting completion...");
    service.shutdown();
    while (!service.isTerminated()) {
        Thread.onSpinWait();
    }
    log.info("All tasks completed.");
}
```

3) Reduce: The implementation of the reduction version of the parallel algorithm displays a similar pattern to the above implementation.

We find again the same orchestrating method...

Listing 8: ParallelReduceMatrixMultiplication#*execute*

```
public Matrix execute() {
    log.info("Executing multiplication...");

    chronometer.start();

    generateAndExecuteTasks();
    awaitCompletion();

    chronometer.stop();
    log.info("Duration: {} ms", String.format("%,d", chronometer.getDuration()));

    return result;
}
```

and two companion methods:

Listing 9: ParallelReduceMatrixMultiplication#*generateAndExecuteTasks*

```
private void generateAndExecuteTasks() {
    log.info("Generating and executing parallel tasks...");
    for (int row = 0; row < height; row++) {
        for (int column = 0; column < width; column++) {
            values[row][column] = 0;
            for (int stripe = 0; stripe < dimension;
                 stripe++) {
                final Task task = new Task(this, row,
                     column,stripe);
                service.execute(task);
            }
        }
    }
    log.info("All tasks generated and completed.");
}
```

Listing 10: ParallelReduceMatrixMultiplication#*awaitCompletion*

```
private void awaitCompletion() {
    log.info("Awaiting completion...");
    service.shutdown();
    while (!service.isTerminated()) {
        Thread.onSpinWait();
    }
    chronometer.stop();
    log.info("All tasks completed.");
}
```

The first method generates and starts all the individual tasks for each of the multiplication operations, whilst the second acts as a join section and patiently awaits the completion of all the tasks.

> As you can easily imagine, this method leads to the generation of a great deal of `Runnable` tasks. Given that a dense matrix will generate millions of micro-tasks, this will quickly lead to a heap overflow. In the very least, it will cause the Garbage Collector (GC) to constantly operate to remove these tasks from the heap, thus crippling the performance of the program. We will observe this in our tests.

4) Stream: Our last implementation takes advantage of the Java Stream API to iterate over the collection of tasks (Java Platform SE 8 2022). It resembles to the 2nd implementation (*stripe parallelism*), to the notable difference that it starts from the stripe.

Listing 11: ParallelStreamMatrixMultiplication#*execute*

```
public Matrix execute() {
    log.info("Executing multiplication...");
    chronometer.start();
    IntStream.range(0, dimension)
          .forEach(this::processStripe);
    chronometer.stop();
    log.info("Duration: {} ms", String.format("%,d",
         chronometer.getDuration()));
    return matrix;
}
```

Listing 12: ParallelStreamMatrixMultiplication#*processStripe*

```
private void processStripe(final int stripe) {
    // Within selected column, process parallel
         multiplication per row.
    IntStream.range(0, height)
          .parallel()
          .forEach(row -> processStripe(row, stripe));
}
```

Listing 13: ParallelStreamMatrixMultiplication#*processStripe*

```
private void processStripe(final int row, final int stripe)
     {
    // For selected row and column in matrix A, compute
         multiplication against entire row in matrix B.
    IntStream.range(0, width)
          .parallel()
          .forEach(column -> multiply(row, stripe, column));
}
```

Listing 14: ParallelStreamMatrixMultiplication#*multiply*

```
private void multiply(final int row, final int stripe,
     final int column) {

    final int valueA = matrixA.get(row, stripe);
    final int valueB = matrixB.get(stripe, column);
    final int product = valueA * valueB;

    values[row][column] += product;
}
```

The first method divides the task space into stripes to be processed sequentially. In the second method, an initial set of threads are started to process each row. In turn, in the third methods, these threads also generate a number of threads to process each column. Finally, in the four method, each sub-thread execute the multiplication operation and updates the product matrix.

Since each index of all stripes is processed sequentially, it guarantees that no thread will ever race with another thread to access the same cell in the product matrix.

C. Tests

In order to assess the performance of our implementations, we ran a mini-benchmark: we defined four set of matrices of various density, then measured the time of execution of five runs.

> We took action to measures the runs after a few others to account for the CPU wind-up and excluded the outliers (which may have been influenced externally).

The results are recorded in the table below.

Matrices		*Approximate Execution Time (ms)*			
		Sequential	**Stripe**	**Reduction**	**Stream**
simple	$[3,3] \times [3,1]$	0.012	0.508	0.609	0.303
medium	$[10,12] \times [12,3]$	0.037	1.145	1.135	1.466
large	$[200,200] \times [200,100]$	3.562	12.642	828.396	36.772
humongous	$[200,2000] \times [2000,100]$	52.396	80.431	13,978.421	464.939
gigantic	$[2000,200] \times [200,1000]$	343.371	404.477	✖	847.089
nightmarish	$[2000,2000] \times [2000,2000]$	30,481.248	16,049.637	✖	21,586.377

Table I: Java Implementation Benchmark Tests

From this table, we can make a number of observations:

- The sequential algorithm performs better on small matrices. However, parallel implementations progressively pace up and take on as the density of the matrices increases.
- Interestingly, the stream implementation hits an outlying high when processing the *humongous* matrices (matrices with a few contention points but a high number of values to process in each stripe.
- As feared, the reduction algorithm does not hold true to this observation. Its initial performance is better but, as matrices get denser, becomes more and more crippled. It even fails when processing the largest matrices because of heap overflow, due to the number of concurrent tasks to process.

THE C PROGRAMMING LANGUAGE

IV. C

We will now discuss our implementation of the matrix multiplication algorithms in C language.

We will start by describing the tree structure of our C files, and introduce you to the most important C headers. After, we will present the two algorithm implementations using OpenMP, and detail the steps involved in the computation. Last, we will present the result of our benchmarking.

A. General Design

Our C program is divided into three directories:

- `main` - the main function used to start the program and execute the matrix tests.
- **`matrix`** contains all the files pertaining to matrices, from IO to representation and multiplication.
  - `matrix`
    The file that controls the internal representation of a matrix, and advertises a number of matrix-related methods.
  - `loader`
    The file that manages the loading of matrix `.mtx` files into memory for computation.
  - `writer`
    The file that operates the writing of a live matrix into a matrix `.mtx` file.
  - `multiply`
    The file that performs the core feature of the program: multiplying matrices.
- **`utils`** contains all the files with an accessory function to the main feature of the program.
  - $\texttt{chrono}_u til$
    A simple utility to measure the lapsing of time between two events.
  - $\texttt{file}_u til$
    A utility for managing files and related pointers.
  - $\texttt{log}_u til$
    An important utility for printing and writing controlled log messages.
  - $\texttt{str}_u til$
    A cryptical utility for concatenating strings on the fly with variadic arguments.
- **`exception`**
  - `exception`
    An aggregating file for listing and tracking exceptional states in the program.

1) Matrix:

Listing 15: matrix.h

```c
#ifndef DCU_MATRIX_H
#define DCU_MATRIX_H

#include <stdlib.h>
#include <stdio.h>
#include "unistd.h"

#include <omp.h>

#include "../utils/log_util.h"
#include "../exceptions/exception.h"

#define MATRIX_FILE_EXT "mtx"

/**
 * The structure representing a matrix in memory.
 */
typedef struct Matrix {
    int **values;
    char *name;
    unsigned int height;
    unsigned int width;
} Matrix;

/**
 * Return a new matrix.
 * @param name the name of the matrix to instantiate.
 * @return a new matrix instance.
 */
Matrix *create_empty_matrix(char *name);

/**
 * Allocate heap memory to a given matrix
 * @param matrix the matrix for which allocate memory.
 * @throw INVALID_MATRIX_DIMENSION if the matrix has any
     dimension equal to 0.
 */
void allocate_memory(Matrix *matrix);

/**
 * Returns a new matrix with allocated memory.
 * @param name the name of the matrix to create.
 * @param height the height of the matrix to create.
 * @param width the width of the matrix to create.
 * @return a new instance of a matrix with a [height,width]
     heap memory allocation.
 * @throw INVALID_MATRIX_DIMENSION if the matrix has any
     dimension equal to 0.
 */
Matrix *create_matrix(char *name, unsigned int height,
    unsigned int width);

/**
 * Free and remove the given matrix from the heap.
 * @param matrix the matrix to destroy.
 */
void destroy_matrix(Matrix *matrix);

/**
 * Display the given matrix on screen.
 * @param matrix the matrix to display.
 */
void display_matrix(Matrix *matrix);

/**
 * Check if two matrices are equal re. their data.
 * @param A the first matrix to compare.
 * @param B the second matrix to compare.
 * @return true if both matrices contain the same values.
 * @throw INVALID_MATRIX_DIMENSION if the matrices'
     dimensions do not match.
 */
bool matrices_are_equal(Matrix *A, Matrix *B);

/**
 * Return the name of the file corresponding to the matrix.
 * @param name the simple matrix name.
 * @return a file name with a matrix extension.
 */
char *get_matrix_file_name(char *name);

#endif //DCU_MATRIX_H
```

As for our Java program, we chose to use a bi-dimensional array to represent matrices in C memory, using a structure. Consequently, it may benefit from the same advantages and suffer from the same integer size limitations.

2) Loader and Writer:

Listing 16: loader.h

```c
#ifndef DCU_MTX_LOADER_H
#define DCU_MTX_LOADER_H

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

#include "matrix.h"
#include "../exceptions/exception.h"

#include "../utils/log_util.h"
#include "../utils/str_util.h"
#include "../utils/file_util.h"

#define MAX_LINE_LENGTH 20000

/**
 * Load a given matrix from a MTX file into memory.
 * @param matrix_name the name of the matrix to load.
 * @return a new instance of matrix.
 * @throw INVALID_MATRIX_DIMENSION if the matrix contained
     in the file is not valid
 *        (any null dimensions/inconsistent row length)
 * @throw FILE_NOT_FOUND if the file is not found (or any
     other IO exception).
 */
Matrix *load_matrix(char *matrix_name);

#endif //DCU_MTX_LOADER_H
```

Listing 17: writer.h

```c
#ifndef DCU_WRITER_H
#define DCU_WRITER_H

#include "matrix.h"

#include "../utils/file_util.h"
#include "../utils/log_util.h"

/**
 * Offload the given matrix to a MTX file on disk.
 * @param matrix the matrix to write.
 * @throw FILE_NOT_FOUND if the file could not be created.
 */
void write_matrix(Matrix *matrix);

#endif //DCU_WRITER_H
```

These two files have a simple function of IO on matrices. Notably, the loading of a matrix of unknown size line by line required to reserve a significant memory space (78 KB) to be able to read the densest matrices. A better implementation would read the file character by character instead.

3) Multiply:

Listing 18: multiply.h

```
#ifndef DCU_MTX_MULTIPLICATION_H
#define DCU_MTX_MULTIPLICATION_H

#include "matrix.h"

#include "../utils/chrono_util.h"

/**
 * Multiply two matrices using a plain sequential algorithm.
 * @param A the left operand matrix.
 * @param B the right operand matrix.
 * @return a new instance of matrix of dimension [A.height,
     B.width] containing the dot product of all vectors.
 * @throw INVALID_MATRIX_DIMENSION if the matrices cannot
     be multiplied because A.width is not equal to B.height.
 */
Matrix *multiply_sequential(Matrix *A, Matrix *B);

/**
 * Multiply two matrices using a stripe parallel algorithm.
 * @param A the left operand matrix.
 * @param B the right operand matrix.
 * @return a new instance of matrix of dimension [A.height,
     B.width] containing the dot product of all vectors.
 * @throw INVALID_MATRIX_DIMENSION if the matrices cannot
     be multiplied because A.width is not equal to B.height.
 */
Matrix *multiply_parallel(Matrix *A, Matrix *B);

#endif //DCU_MTX_MULTIPLICATION_H
```

In this file are advertised two implementations of the matrix multiplication algorithms.

B. Implementation

In our C program, we have restricted our implementations of the matrix multiplication to two algorithms (compared to four in Java).

- Because C does not offer a `stream` API out of the box.
- Because operating mutexes in OpenMP was not practical.

The algorithms implemented are those exposed in the `multiply` header:

- `multiply_sequential` implements the sequential algorithm.
- `multiply_parallel` implements the stripe parallel algorithm.

1) Sequential: The sequential algorithm implementation is strictly identical to that of Java.

Listing 19: multiply#*multiply_sequential*

```
int **values = product->values;
for (int row = 0; row < A->height; ++row) {
    for (int column = 0; column < B->width; ++column) {
        values[row][column] = 0;
        for (int stripe = 0; stripe < A->width; stripe++) {
            values[row][column] += A->values[row][stripe] * B->values[stripe][column];
        }
    }
}
```

A simple imbrication of `for` loop updating the values of the product matrix in increasing order.

2) Parallel: The parallel implementation makes use of the OpenMP library to parallelize the processing of each stripe.

Listing 20: multiply#*multiply_parallel*

```
#pragma omp parallel for
for (int row = 0; row < A->height; ++row) {

#pragma omp parallel for
    for (int column = 0; column < B->width; ++column) {
        process_stripe_parallel(C, A, B, row, column);
    }
}
```

Listing 21: multiply#*process_stripe_parallel*

```
int sum = 0;
#pragma omp parallel for reduction (+:sum)
for (int stripe = 0; stripe < A->width; stripe++) {
    sum += A->values[row][stripe] *
         B->values[stripe][column];
}
product->values[row][column] = sum;
```

As you can see, we have used three *pragma* in this implementation:

- The first two *pragma* create a set of threads corresponding to each $\langle row, column \rangle$ pair.
- The last *pragma* create a set of thread to iterate over a stripe: the multiplication operations are treated concurrently, while the accumulation is handled by the reduction operation.

C. Tests

To quantify the performance of each algorithm, we have again made a micro-benchmarking of 5 runs against the same matrices as the Java program.

The results are indicated below.

Matrices		*Approximate Execution Time (ms)*	
		Sequential	**Parallel**
simple	$[3, 3] \times [3, 1]$	0.001	0.169
medium	$[10, 12] \times [12, 3]$	0.002	0.227
large	$[200, 200] \times [200, 100]$	13.378	4.214
humongous	$[200, 2000] \times [2000, 100]$	165.184	29.041
gigantic	$[2000, 200] \times [200, 1000]$	1,652.985	576.619
nightmarish	$[2000, 2000] \times [2000, 2000]$	42,910.816	14,497.616

Table II: C Implementation Benchmark Tests

In opposition to Java, the C parallel implementation of the matrix multiplication quickly outperforms the sequential implementation. With a matrix as small as $[200, 200]$, it already becomes 3 times faster!

V. Discussion

We will now review the performance of the implementation in both Java and C. We will start by aggregating the results into a single table, and mark which language and implementation fare better against a set of matrices. Doing so, we will try to analyse the order of performance and compare our assumptions with the actual results. We will then conclude this document by offering our opinion on the realization of this assignment.

A. Commentary

Matrices	Java				C	
	Approximate Execution Time (ms)					
	Sequential	**Stripe**	**Reduction**	**Stream**	**Sequential**	**Parallel**
$[3,3] \times [3,1]$	0.012 (2)	0.508	0.609	0.303	0.001 (1)	0.169 (3)
$[10,12] \times [12,3]$	0.037 (2)	1.145	1.135 (3)	1.466	0.002 (1)	0.227
$[200,200] \times [200,100]$	3.562 (1)	12.642 (3)	828.396	36.772	13.378	4.214 (2)
$[200,2000] \times [2000,100]$	52.396 (2)	80.431 (3)	13,978.421	464.939	165.184	29.041 (1)
$[2000,200] \times [200,1000]$	343.371 (1)	404.477 (2)	✖	847.089	1,652.985	576.619
$[2000,2000] \times [2000,2000]$	30,481.248	16,049.637 (2)	✖	21,586.377 (3)	42,910.816	14,497.616 (1)

Table III: Java/C Implementation Benchmark Tests

As expected, the performance of the sequential implementations decreases as the density of the matrix increases, as compared to parallel algorithms, and the C-based implementation have a higher throughput on higher density. However, a few surprises came out of the table.

First of all, the Java sequential algorithm is strongly resilient, and it is only after density hit the high mark of $[2000, 2000]$ that it concedes performance. What is even more surprising is that it frequently outperforms its C equivalent, even though they both implement the same `for` loops. We presume that this may be the act of the Java Compiler and JVM, which may introduce loop optimizations into play (loop unrolling, CPU pipelining, ...).

Second, parallel algorithms seems to gain ground whenever the size of the stripes increases. This is actually very logical,

- Since a large count of stripes will create many threads that cannot be handled parallelly by the CPU, these will be queued for execution. This has the same effect as sequential processing.
- When several threads run concurrently, the higher the size of the stripe, the more gain they will offer against a mono-thread executing all instructions sequentially.

Third, the performance of the Java Streams decreases over the increase in density, compared to the regular stripe implementation. We suppose that this, in opposition to the `for` loops, is caused the lack of optimization (though this would be surprising). The Stripe implementation, on the other hand, achieves good grades overall.

Finally, and this one is not a surprise, the performance of the reduction algorithm is very disappointing. However, this is easily explained by the explosion of instances of `Task`. The tradeoff between the overhead and memory consumption of a task versus the instantaneous operation it is representing is probably not worth it. Moreover, the concurrency over the locks makes it such that many threads may be blocked and consume memory. All this while the lock is theoretically released, either because the locking thread has completed the operation but was suspended before releasing the lock, or because the lock is released but all the concerned threads are currently queued behind other threads.

We will also add that C seems to be performing better in high parallelism, which we attribute to the lower level of programming associated with such programs. Java may indeed be influenced by its object-oriented paradigm and JVM architecture, and suffer from overhead from this point of view.

B. Conclusion

Contrary to our belief, it was not harder to implement concurrency in C compared to Java. Actually, the OpenMP pragma and for loops were easier to develop and involved very limited errors. Of course, this was influenced by the low complexity of our algorithms, and it is not sure that this would hold true on more advanced algorithm needs. Nevertheless, its high performance and low complexity makes it a good candidate for implementing performance-sensitive applications requiring a significant level of parallelism.

References

Maths Resources

Alwis, Roshan (Aug. 2017). "Parallel Matrix Multiplication [C][Parallel Processing]". In: *Tech Vision*. medium. URL: https://medium.com/tech-vision/parallel-matrix-multiplication-c-parallel-processing-5e3aadb36f27 (visited on 03/12/2022).

Gergel, V.P (2005). "Parallel Methods for Matrix Multiplication". In: *Introduction to Parallel Programming*. 8. Faculty of Computational Mathematics & Cybernetics. University of Nizhni Novgorod. URL: http://www.lac.inpe.br/~stephan/CAP-372/matrixmult_microsoft.pdf (visited on 03/12/2022).

Pierce, Rod (2021). *How to Multiply Matrices*. Math is Fun. URL: https://www.mathsisfun.com/algebra/matrix-multiplying.html (visited on 03/15/2022).

Wikipedia (2022). *Matrix (Mathematics)*. Wikimedia, Inc. URL: https://en.wikipedia.org/wiki/Matrix_(mathematics) (visited on 03/15/2022).

Image Credits

juicy_fish (2022a). *Loans Soft-fill*. Flaticon. URL: https://www.flaticon.com/packs/loans-2 (visited on 03/16/2022).

– (2022b). *Trophies And Awards — Soft-fill*. Flaticon. URL: https://www.flaticon.com/packs/trophies-and-awards-3 (visited on 03/16/2022).

Svjo (2022). *Matrix multiplication*. Wikimedia, Inc. URL: https://en.wikipedia.org/wiki/Matrix_(mathematics) (visited on 03/15/2022).

Code resources

function strtok (2021). cplusplus.com. URL: https://www.cplusplus.com/reference/cstring/strtok/ (visited on 03/13/2022).

Ghosh, Bamdeb (3). *How to use gettimeofday function in C language?* linuxhint. URL: https://linuxhint.com/gettimeofday_c_language/ (visited on 03/15/2022).

Java Platform SE 8 (2022). *Interface Stream<T>*. Oracle. URL: https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html (visited on 03/15/2022).

Neto, Silveira (3). *Simple Java Chronometer*. URL: http://silveiraneto.net/2008/03/15/simple-java-chronometer/ (visited on 03/15/2022).

Other

Generate Random Matrices (2022). Online Math Tools. URL: https://onlinemathtools.com/generate-random-matrix (visited on 03/16/2022).

Annexes

Java Logs

Listing 22: large–sequential.log

```
2022-03-16 11:57:52,015 INFO  [main] o.d.s.s.c.a.MultiplicationTestByMatrix    :23  - Performing Matrices Multiplication Test...
2022-03-16 11:57:52,020 WARN  [main] o.d.s.s.c.a.MultiplicationTestByMatrix    :24  - Selected algorithm: Sequential
2022-03-16 11:57:52,022 INFO  [main] o.d.s.s.c.a.MultiplicationTestByMatrix    :84  - Setting up...
2022-03-16 11:57:52,027 INFO  [main] o.d.s.s.c.m.MatrixLoader                  :48  - Loading new matrix [largeA]...
2022-03-16 11:57:52,028 INFO  [main] o.d.s.s.c.m.Matrix                        :47  - Creating empty matrix...
2022-03-16 11:57:52,029 INFO  [main] o.d.s.s.c.m.MatrixLoader                  :69  - Loading rows from file [largeA.mtx]...
2022-03-16 11:57:52,062 INFO  [main] o.d.s.s.c.m.MatrixLoader                  :79  - Rows loaded.
2022-03-16 11:57:52,062 INFO  [main] o.d.s.s.c.m.MatrixLoader                  :95  - Estimating memory need...
2022-03-16 11:57:52,064 INFO  [main] o.d.s.s.c.m.MatrixLoader                  :101 - Memory needs: [200,200]
2022-03-16 11:57:52,065 INFO  [main] o.d.s.s.c.m.Matrix                        :56  - Allocating memory for [200,200] matrix...
2022-03-16 11:57:52,068 INFO  [main] o.d.s.s.c.m.Matrix                        :64  - Memory allocated.
2022-03-16 11:57:52,069 INFO  [main] o.d.s.s.c.m.Matrix                        :83  - Filling [200,200] matrix...
2022-03-16 11:57:52,069 INFO  [main] o.d.s.s.c.m.Matrix                        :84  - Copying matrix's file content...
2022-03-16 11:57:52,076 INFO  [main] o.d.s.s.c.m.Matrix                        :91  - Matrix's file content copied.
2022-03-16 11:57:52,076 INFO  [main] o.d.s.s.c.m.MatrixLoader                  :48  - Loading new matrix [largeB]...
2022-03-16 11:57:52,077 INFO  [main] o.d.s.s.c.m.Matrix                        :47  - Creating empty matrix...
2022-03-16 11:57:52,077 INFO  [main] o.d.s.s.c.m.MatrixLoader                  :69  - Loading rows from file [largeB.mtx]...
2022-03-16 11:57:52,081 INFO  [main] o.d.s.s.c.m.MatrixLoader                  :79  - Rows loaded.
2022-03-16 11:57:52,082 INFO  [main] o.d.s.s.c.m.MatrixLoader                  :95  - Estimating memory need...
2022-03-16 11:57:52,082 INFO  [main] o.d.s.s.c.m.MatrixLoader                  :101 - Memory needs: [200,100]
2022-03-16 11:57:52,082 INFO  [main] o.d.s.s.c.m.Matrix                        :56  - Allocating memory for [200,100] matrix...
2022-03-16 11:57:52,083 INFO  [main] o.d.s.s.c.m.Matrix                        :64  - Memory allocated.
2022-03-16 11:57:52,083 INFO  [main] o.d.s.s.c.m.Matrix                        :83  - Filling [200,100] matrix...
2022-03-16 11:57:52,083 INFO  [main] o.d.s.s.c.m.Matrix                        :84  - Copying matrix's file content...
2022-03-16 11:57:52,084 INFO  [main] o.d.s.s.c.m.Matrix                        :91  - Matrix's file content copied.
2022-03-16 11:57:52,085 INFO  [main] o.d.s.s.c.a.MultiplicationTestByMatrix    :88  - Performing test...
2022-03-16 11:57:52,085 INFO  [main] o.d.s.s.c.a.Algorithm                     :58  - Computing product of largeA and largeB...
2022-03-16 11:57:52,086 INFO  [main] o.d.s.s.c.a.SequentialMatrixMultiplication:13  - Using Sequential method...
2022-03-16 11:57:52,093 INFO  [main] o.d.s.s.c.m.Matrix                        :47  - Creating empty matrix...
2022-03-16 11:57:52,093 INFO  [main] o.d.s.s.c.m.Matrix                        :56  - Allocating memory for [200,100] matrix...
2022-03-16 11:57:52,094 INFO  [main] o.d.s.s.c.m.Matrix                        :64  - Memory allocated.
2022-03-16 11:57:52,094 INFO  [main] o.d.s.s.c.a.SequentialMatrixMultiplication:29  - Executing multiplication...
2022-03-16 11:57:52,094 INFO  [main] o.d.s.s.c.u.Chronometer                   :33  - Creating new chronometer...
2022-03-16 11:57:52,095 INFO  [main] o.d.s.s.c.u.Chronometer                   :34  - Chronometer created.
2022-03-16 11:57:52,095 INFO  [main] o.d.s.s.c.u.Chronometer                   :66  - Starting chronometer...
2022-03-16 11:57:52,098 INFO  [main] o.d.s.s.c.u.Chronometer                   :74  - Chronometer started.
2022-03-16 11:57:52,121 INFO  [main] o.d.s.s.c.u.Chronometer                   :86  - Stopping chronometer...
2022-03-16 11:57:52,121 INFO  [main] o.d.s.s.c.u.Chronometer                   :97  - Chronometer stopped.
2022-03-16 11:57:52,123 WARN  [main] o.d.s.s.c.a.SequentialMatrixMultiplication:42  - Duration: 24.935 ms
2022-03-16 11:57:52,170 INFO  [main] o.d.s.s.c.a.MultiplicationTestByMatrix    :92  - Validating result...
2022-03-16 11:57:52,171 INFO  [main] o.d.s.s.c.m.MatrixLoader                  :48  - Loading new matrix [largeC]...
2022-03-16 11:57:52,171 INFO  [main] o.d.s.s.c.m.Matrix                        :47  - Creating empty matrix...
2022-03-16 11:57:52,171 INFO  [main] o.d.s.s.c.m.MatrixLoader                  :69  - Loading rows from file [largeC.mtx]...
2022-03-16 11:57:52,176 INFO  [main] o.d.s.s.c.m.MatrixLoader                  :79  - Rows loaded.
2022-03-16 11:57:52,176 INFO  [main] o.d.s.s.c.m.MatrixLoader                  :95  - Estimating memory need...
2022-03-16 11:57:52,176 INFO  [main] o.d.s.s.c.m.MatrixLoader                  :101 - Memory needs: [200,100]
2022-03-16 11:57:52,176 INFO  [main] o.d.s.s.c.m.Matrix                        :56  - Allocating memory for [200,100] matrix...
2022-03-16 11:57:52,176 INFO  [main] o.d.s.s.c.m.Matrix                        :64  - Memory allocated.
2022-03-16 11:57:52,177 INFO  [main] o.d.s.s.c.m.Matrix                        :83  - Filling [200,100] matrix...
2022-03-16 11:57:52,177 INFO  [main] o.d.s.s.c.m.Matrix                        :84  - Copying matrix's file content...
2022-03-16 11:57:52,178 INFO  [main] o.d.s.s.c.m.Matrix                        :91  - Matrix's file content copied.
2022-03-16 11:57:52,178 INFO  [main] o.d.s.s.c.m.Matrix                        :140 - Comparing matrix [largeAxlargeB] with matrix [largeC]
2022-03-16 11:57:52,179 INFO  [main] o.d.s.s.c.m.Matrix                        :153 - Matrices match.
```

Listing 23: large–stripe.log

```
2022-03-16 11:57:52,181 INFO  [main] o.d.s.s.c.a.MultiplicationTestByMatrix    :23  - Performing Matrices Multiplication Test...
2022-03-16 11:57:52,181 WARN  [main] o.d.s.s.c.a.MultiplicationTestByMatrix    :24  - Selected algorithm: Parallel Striping
2022-03-16 11:57:52,181 INFO  [main] o.d.s.s.c.a.MultiplicationTestByMatrix    :84  - Setting up...
2022-03-16 11:57:52,181 INFO  [main] o.d.s.s.c.m.MatrixLoader                  :48  - Loading new matrix [largeA]...
2022-03-16 11:57:52,182 INFO  [main] o.d.s.s.c.m.Matrix                        :47  - Creating empty matrix...
2022-03-16 11:57:52,182 INFO  [main] o.d.s.s.c.m.MatrixLoader                  :69  - Loading rows from file [largeA.mtx]...
2022-03-16 11:57:52,187 INFO  [main] o.d.s.s.c.m.MatrixLoader                  :79  - Rows loaded.
2022-03-16 11:57:52,188 INFO  [main] o.d.s.s.c.m.MatrixLoader                  :95  - Estimating memory need...
2022-03-16 11:57:52,188 INFO  [main] o.d.s.s.c.m.MatrixLoader                  :101 - Memory needs: [200,200]
2022-03-16 11:57:52,188 INFO  [main] o.d.s.s.c.m.Matrix                        :56  - Allocating memory for [200,200] matrix...
2022-03-16 11:57:52,188 INFO  [main] o.d.s.s.c.m.Matrix                        :64  - Memory allocated.
2022-03-16 11:57:52,188 INFO  [main] o.d.s.s.c.m.Matrix                        :83  - Filling [200,200] matrix...
2022-03-16 11:57:52,188 INFO  [main] o.d.s.s.c.m.Matrix                        :84  - Copying matrix's file content...
2022-03-16 11:57:52,190 INFO  [main] o.d.s.s.c.m.Matrix                        :91  - Matrix's file content copied.
2022-03-16 11:57:52,190 INFO  [main] o.d.s.s.c.m.MatrixLoader                  :48  - Loading new matrix [largeB]...
2022-03-16 11:57:52,190 INFO  [main] o.d.s.s.c.m.Matrix                        :47  - Creating empty matrix...
2022-03-16 11:57:52,191 INFO  [main] o.d.s.s.c.m.MatrixLoader                  :69  - Loading rows from file [largeB.mtx]...
2022-03-16 11:57:52,193 INFO  [main] o.d.s.s.c.m.MatrixLoader                  :79  - Rows loaded.
2022-03-16 11:57:52,194 INFO  [main] o.d.s.s.c.m.MatrixLoader                  :95  - Estimating memory need...
2022-03-16 11:57:52,194 INFO  [main] o.d.s.s.c.m.MatrixLoader                  :101 - Memory needs: [200,100]
2022-03-16 11:57:52,194 INFO  [main] o.d.s.s.c.m.Matrix                        :56  - Allocating memory for [200,100] matrix...
2022-03-16 11:57:52,194 INFO  [main] o.d.s.s.c.m.Matrix                        :64  - Memory allocated.
2022-03-16 11:57:52,195 INFO  [main] o.d.s.s.c.m.Matrix                        :83  - Filling [200,100] matrix...
2022-03-16 11:57:52,195 INFO  [main] o.d.s.s.c.m.Matrix                        :84  - Copying matrix's file content...
2022-03-16 11:57:52,195 INFO  [main] o.d.s.s.c.m.Matrix                        :91  - Matrix's file content copied.
2022-03-16 11:57:52,195 INFO  [main] o.d.s.s.c.a.MultiplicationTestByMatrix    :88  - Performing test...
2022-03-16 11:57:52,196 INFO  [main] o.d.s.s.c.a.Algorithm                     :58  - Computing product of largeA and largeB...
2022-03-16 11:57:52,196 INFO  [main] o.d.s.s.c.a.ParallelStripeMatrixMultiplication:19  - Using Parallel Reduction method...
2022-03-16 11:57:52,198 INFO  [main] o.d.s.s.c.u.Chronometer                   :33  - Creating new chronometer...
2022-03-16 11:57:52,198 INFO  [main] o.d.s.s.c.u.Chronometer                   :34  - Chronometer created.
2022-03-16 11:57:52,199 INFO  [main] o.d.s.s.c.a.ParallelStripeMatrixMultiplication:41  - Creating context...
2022-03-16 11:57:52,199 INFO  [main] o.d.s.s.c.m.Matrix                        :47  - Creating empty matrix...
2022-03-16 11:57:52,199 INFO  [main] o.d.s.s.c.m.Matrix                        :56  - Allocating memory for [200,100] matrix...
2022-03-16 11:57:52,199 INFO  [main] o.d.s.s.c.m.Matrix                        :64  - Memory allocated.
2022-03-16 11:57:52,199 INFO  [main] o.d.s.s.c.a.ParallelStripeMatrixMultiplication:59  - Executing multiplication...
2022-03-16 11:57:52,199 INFO  [main] o.d.s.s.c.u.Chronometer                   :66  - Starting chronometer...
2022-03-16 11:57:52,199 INFO  [main] o.d.s.s.c.u.Chronometer                   :74  - Chronometer started.
2022-03-16 11:57:52,199 INFO  [main] o.d.s.s.c.a.ParallelStripeMatrixMultiplication:74  - Generating and executing tasks...
2022-03-16 11:57:52,219 INFO  [main] o.d.s.s.c.a.ParallelStripeMatrixMultiplication:81  - Tasks generated.
2022-03-16 11:57:52,219 INFO  [main] o.d.s.s.c.a.ParallelStripeMatrixMultiplication:85  - Awaiting completion...
2022-03-16 11:57:52,287 INFO  [main] o.d.s.s.c.a.ParallelStripeMatrixMultiplication:90  - All tasks completed.
2022-03-16 11:57:52,287 INFO  [main] o.d.s.s.c.u.Chronometer                   :86  - Stopping chronometer...
2022-03-16 11:57:52,287 INFO  [main] o.d.s.s.c.u.Chronometer                   :97  - Chronometer stopped.
2022-03-16 11:57:52,287 WARN  [main] o.d.s.s.c.a.ParallelStripeMatrixMultiplication:68  - Duration: 87.738 ms
2022-03-16 11:57:52,288 INFO  [main] o.d.s.s.c.a.MultiplicationTestByMatrix    :92  - Validating result...
2022-03-16 11:57:52,288 INFO  [main] o.d.s.s.c.m.MatrixLoader                  :48  - Loading new matrix [largeC]...
2022-03-16 11:57:52,288 INFO  [main] o.d.s.s.c.m.Matrix                        :47  - Creating empty matrix...
2022-03-16 11:57:52,288 INFO  [main] o.d.s.s.c.m.MatrixLoader                  :69  - Loading rows from file [largeC.mtx]...
2022-03-16 11:57:52,292 INFO  [main] o.d.s.s.c.m.MatrixLoader                  :79  - Rows loaded.
2022-03-16 11:57:52,292 INFO  [main] o.d.s.s.c.m.MatrixLoader                  :95  - Estimating memory need...
2022-03-16 11:57:52,292 INFO  [main] o.d.s.s.c.m.MatrixLoader                  :101 - Memory needs: [200,100]
2022-03-16 11:57:52,292 INFO  [main] o.d.s.s.c.m.Matrix                        :56  - Allocating memory for [200,100] matrix...
2022-03-16 11:57:52,292 INFO  [main] o.d.s.s.c.m.Matrix                        :64  - Memory allocated.
2022-03-16 11:57:52,293 INFO  [main] o.d.s.s.c.m.Matrix                        :83  - Filling [200,100] matrix...
2022-03-16 11:57:52,293 INFO  [main] o.d.s.s.c.m.Matrix                        :84  - Copying matrix's file content...
2022-03-16 11:57:52,294 INFO  [main] o.d.s.s.c.m.Matrix                        :91  - Matrix's file content copied.
2022-03-16 11:57:52,294 INFO  [main] o.d.s.s.c.m.Matrix                        :140 - Comparing matrix [largeAxlargeB] with matrix [largeC]
2022-03-16 11:57:52,294 INFO  [main] o.d.s.s.c.m.Matrix                        :153 - Matrices match.
```

Listing 24: large–stream.log

```
2022-03-16 11:57:52,295 INFO  [main] o.d.s.s.c.a.MultiplicationTestByMatrix    :23  - Performing Matrices Multiplication Test...
2022-03-16 11:57:52,295 WARN  [main] o.d.s.s.c.a.MultiplicationTestByMatrix    :24  - Selected algorithm: Java Streams parallelism
2022-03-16 11:57:52,296 INFO  [main] o.d.s.s.c.a.MultiplicationTestByMatrix    :84  - Setting up...
2022-03-16 11:57:52,298 INFO  [main] o.d.s.s.c.m.MatrixLoader                 :48  - Loading new matrix [largeA]...
2022-03-16 11:57:52,298 INFO  [main] o.d.s.s.c.m.Matrix                       :47  - Creating empty matrix...
2022-03-16 11:57:52,298 INFO  [main] o.d.s.s.c.m.MatrixLoader                 :69  - Loading rows from file [largeA.mtx]...
2022-03-16 11:57:52,303 INFO  [main] o.d.s.s.c.m.MatrixLoader                 :79  - Rows loaded.
2022-03-16 11:57:52,303 INFO  [main] o.d.s.s.c.m.MatrixLoader                 :95  - Estimating memory need...
2022-03-16 11:57:52,304 INFO  [main] o.d.s.s.c.m.MatrixLoader                 :101 - Memory needs: [200,200]
2022-03-16 11:57:52,304 INFO  [main] o.d.s.s.c.m.Matrix                       :56  - Allocating memory for [200,200] matrix...
2022-03-16 11:57:52,304 INFO  [main] o.d.s.s.c.m.Matrix                       :64  - Memory allocated.
2022-03-16 11:57:52,304 INFO  [main] o.d.s.s.c.m.Matrix                       :83  - Filling [200,200] matrix...
2022-03-16 11:57:52,304 INFO  [main] o.d.s.s.c.m.Matrix                       :84  - Copying matrix's file content...
2022-03-16 11:57:52,305 INFO  [main] o.d.s.s.c.m.Matrix                       :91  - Matrix's file content copied.
2022-03-16 11:57:52,305 INFO  [main] o.d.s.s.c.m.MatrixLoader                 :48  - Loading new matrix [largeB]...
2022-03-16 11:57:52,305 INFO  [main] o.d.s.s.c.m.Matrix                       :47  - Creating empty matrix...
2022-03-16 11:57:52,305 INFO  [main] o.d.s.s.c.m.MatrixLoader                 :69  - Loading rows from file [largeB.mtx]...
2022-03-16 11:57:52,308 INFO  [main] o.d.s.s.c.m.MatrixLoader                 :79  - Rows loaded.
2022-03-16 11:57:52,308 INFO  [main] o.d.s.s.c.m.MatrixLoader                 :95  - Estimating memory need...
2022-03-16 11:57:52,308 INFO  [main] o.d.s.s.c.m.MatrixLoader                 :101 - Memory needs: [200,100]
2022-03-16 11:57:52,308 INFO  [main] o.d.s.s.c.m.Matrix                       :56  - Allocating memory for [200,100] matrix...
2022-03-16 11:57:52,308 INFO  [main] o.d.s.s.c.m.Matrix                       :64  - Memory allocated.
2022-03-16 11:57:52,308 INFO  [main] o.d.s.s.c.m.Matrix                       :83  - Filling [200,100] matrix...
2022-03-16 11:57:52,308 INFO  [main] o.d.s.s.c.m.Matrix                       :84  - Copying matrix's file content...
2022-03-16 11:57:52,309 INFO  [main] o.d.s.s.c.m.Matrix                       :91  - Matrix's file content copied.
2022-03-16 11:57:52,309 INFO  [main] o.d.s.s.c.a.MultiplicationTestByMatrix    :88  - Performing test...
2022-03-16 11:57:52,309 INFO  [main] o.d.s.s.c.a.Algorithm                    :58  - Computing product of largeA and largeB...
2022-03-16 11:57:52,309 INFO  [main] o.d.s.s.c.a.ParallelStreamMatrixMultiplication:15  - Using Parallel Reduction method...
2022-03-16 11:57:52,310 INFO  [main] o.d.s.s.c.u.Chronometer                  :33  - Creating new chronometer...
2022-03-16 11:57:52,310 INFO  [main] o.d.s.s.c.u.Chronometer                  :34  - Chronometer created.
2022-03-16 11:57:52,310 INFO  [main] o.d.s.s.c.a.ParallelStreamMatrixMultiplication:36  - Creating context...
2022-03-16 11:57:52,310 INFO  [main] o.d.s.s.c.m.Matrix                       :47  - Creating empty matrix...
2022-03-16 11:57:52,310 INFO  [main] o.d.s.s.c.m.Matrix                       :56  - Allocating memory for [200,100] matrix...
2022-03-16 11:57:52,310 INFO  [main] o.d.s.s.c.m.Matrix                       :64  - Memory allocated.
2022-03-16 11:57:52,311 INFO  [main] o.d.s.s.c.a.ParallelStreamMatrixMultiplication:52  - Context created.
2022-03-16 11:57:52,311 INFO  [main] o.d.s.s.c.a.ParallelStreamMatrixMultiplication:56  - Executing multiplication...
2022-03-16 11:57:52,311 INFO  [main] o.d.s.s.c.u.Chronometer                  :66  - Starting chronometer...
2022-03-16 11:57:52,311 INFO  [main] o.d.s.s.c.u.Chronometer                  :74  - Chronometer started.
2022-03-16 11:57:52,754 INFO  [main] o.d.s.s.c.u.Chronometer                  :86  - Stopping chronometer...
2022-03-16 11:57:52,755 INFO  [main] o.d.s.s.c.u.Chronometer                  :97  - Chronometer stopped.
2022-03-16 11:57:52,755 WARN  [main] o.d.s.s.c.a.ParallelStreamMatrixMultiplication:61  - Duration: 443.994 ms
2022-03-16 11:57:52,755 INFO  [main] o.d.s.s.c.a.MultiplicationTestByMatrix    :92  - Validating result...
2022-03-16 11:57:52,756 INFO  [main] o.d.s.s.c.m.MatrixLoader                 :48  - Loading new matrix [largeC]...
2022-03-16 11:57:52,756 INFO  [main] o.d.s.s.c.m.Matrix                       :47  - Creating empty matrix...
2022-03-16 11:57:52,756 INFO  [main] o.d.s.s.c.m.MatrixLoader                 :69  - Loading rows from file [largeC.mtx]...
2022-03-16 11:57:52,772 INFO  [main] o.d.s.s.c.m.MatrixLoader                 :79  - Rows loaded.
2022-03-16 11:57:52,772 INFO  [main] o.d.s.s.c.m.MatrixLoader                 :95  - Estimating memory need...
2022-03-16 11:57:52,772 INFO  [main] o.d.s.s.c.m.MatrixLoader                 :101 - Memory needs: [200,100]
2022-03-16 11:57:52,772 INFO  [main] o.d.s.s.c.m.Matrix                       :56  - Allocating memory for [200,100] matrix...
2022-03-16 11:57:52,773 INFO  [main] o.d.s.s.c.m.Matrix                       :64  - Memory allocated.
2022-03-16 11:57:52,773 INFO  [main] o.d.s.s.c.m.Matrix                       :83  - Filling [200,100] matrix...
2022-03-16 11:57:52,773 INFO  [main] o.d.s.s.c.m.Matrix                       :84  - Copying matrix's file content...
2022-03-16 11:57:52,774 INFO  [main] o.d.s.s.c.m.Matrix                       :91  - Matrix's file content copied.
2022-03-16 11:57:52,774 INFO  [main] o.d.s.s.c.m.Matrix                       :140 - Comparing matrix [largeAxlargeB] with matrix [largeC]
2022-03-16 11:57:52,774 INFO  [main] o.d.s.s.c.m.Matrix                       :153 - Matrices match.
```

Listing 25: large–reduce.log

```
2022-03-16 11:57:52,775 INFO  [main] o.d.s.s.c.a.MultiplicationTestByMatrix    :23  - Performing Matrices Multiplication Test...
2022-03-16 11:57:52,775 WARN  [main] o.d.s.s.c.a.MultiplicationTestByMatrix    :24  - Selected algorithm: Parallel Reduction
2022-03-16 11:57:52,775 INFO  [main] o.d.s.s.c.a.MultiplicationTestByMatrix    :84  - Setting up...
2022-03-16 11:57:52,775 INFO  [main] o.d.s.s.c.m.MatrixLoader                  :48  - Loading new matrix [largeA]...
2022-03-16 11:57:52,775 INFO  [main] o.d.s.s.c.m.Matrix                        :47  - Creating empty matrix...
2022-03-16 11:57:52,776 INFO  [main] o.d.s.s.c.m.MatrixLoader                  :69  - Loading rows from file [largeA.mtx]...
2022-03-16 11:57:52,793 INFO  [main] o.d.s.s.c.m.MatrixLoader                  :79  - Rows loaded.
2022-03-16 11:57:52,793 INFO  [main] o.d.s.s.c.m.MatrixLoader                  :95  - Estimating memory need...
2022-03-16 11:57:52,793 INFO  [main] o.d.s.s.c.m.MatrixLoader                  :101 - Memory needs: [200,200]
2022-03-16 11:57:52,793 INFO  [main] o.d.s.s.c.m.Matrix                        :56  - Allocating memory for [200,200] matrix...
2022-03-16 11:57:52,793 INFO  [main] o.d.s.s.c.m.Matrix                        :64  - Memory allocated.
2022-03-16 11:57:52,794 INFO  [main] o.d.s.s.c.m.Matrix                        :83  - Filling [200,200] matrix...
2022-03-16 11:57:52,794 INFO  [main] o.d.s.s.c.m.Matrix                        :84  - Copying matrix's file content...
2022-03-16 11:57:52,794 INFO  [main] o.d.s.s.c.m.Matrix                        :91  - Matrix's file content copied.
2022-03-16 11:57:52,794 INFO  [main] o.d.s.s.c.m.MatrixLoader                  :48  - Loading new matrix [largeB]...
2022-03-16 11:57:52,794 INFO  [main] o.d.s.s.c.m.Matrix                        :47  - Creating empty matrix...
2022-03-16 11:57:52,794 INFO  [main] o.d.s.s.c.m.MatrixLoader                  :69  - Loading rows from file [largeB.mtx]...
2022-03-16 11:57:52,803 INFO  [main] o.d.s.s.c.m.MatrixLoader                  :79  - Rows loaded.
2022-03-16 11:57:52,803 INFO  [main] o.d.s.s.c.m.MatrixLoader                  :95  - Estimating memory need...
2022-03-16 11:57:52,803 INFO  [main] o.d.s.s.c.m.MatrixLoader                  :101 - Memory needs: [200,100]
2022-03-16 11:57:52,803 INFO  [main] o.d.s.s.c.m.Matrix                        :56  - Allocating memory for [200,100] matrix...
2022-03-16 11:57:52,803 INFO  [main] o.d.s.s.c.m.Matrix                        :64  - Memory allocated.
2022-03-16 11:57:52,804 INFO  [main] o.d.s.s.c.m.Matrix                        :83  - Filling [200,100] matrix...
2022-03-16 11:57:52,804 INFO  [main] o.d.s.s.c.m.Matrix                        :84  - Copying matrix's file content...
2022-03-16 11:57:52,804 INFO  [main] o.d.s.s.c.m.Matrix                        :91  - Matrix's file content copied.
2022-03-16 11:57:52,804 INFO  [main] o.d.s.s.c.a.MultiplicationTestByMatrix    :88  - Performing test...
2022-03-16 11:57:52,804 INFO  [main] o.d.s.s.c.a.Algorithm                     :58  - Computing product of largeA and largeB...
2022-03-16 11:57:52,804 INFO  [main] o.d.s.s.c.a.ParallelReduceMatrixMultiplication:21  - Using Parallel Reduction method...
2022-03-16 11:57:52,804 INFO  [main] o.d.s.s.c.u.Chronometer                   :33  - Creating new chronometer...
2022-03-16 11:57:52,805 INFO  [main] o.d.s.s.c.u.Chronometer                   :34  - Chronometer created.
2022-03-16 11:57:52,805 INFO  [main] o.d.s.s.c.a.ParallelReduceMatrixMultiplication:42  - Creating new context...
2022-03-16 11:57:52,807 INFO  [main] o.d.s.s.c.m.Matrix                        :47  - Creating empty matrix...
2022-03-16 11:57:52,807 INFO  [main] o.d.s.s.c.m.Matrix                        :56  - Allocating memory for [200,100] matrix...
2022-03-16 11:57:52,807 INFO  [main] o.d.s.s.c.m.Matrix                        :64  - Memory allocated.
2022-03-16 11:57:52,807 INFO  [main] o.d.s.s.c.a.ParallelReduceMatrixMultiplication:64  - Executing multiplication...
2022-03-16 11:57:52,807 INFO  [main] o.d.s.s.c.u.Chronometer                   :66  - Starting chronometer...
2022-03-16 11:57:52,807 INFO  [main] o.d.s.s.c.u.Chronometer                   :74  - Chronometer started.
2022-03-16 11:57:52,807 INFO  [main] o.d.s.s.c.a.ParallelReduceMatrixMultiplication:80  - Generating and executing parallel tasks...
2022-03-16 11:57:53,646 INFO  [main] o.d.s.s.c.a.ParallelReduceMatrixMultiplication:91  - All tasks generated and completed.
2022-03-16 11:57:53,647 INFO  [main] o.d.s.s.c.a.ParallelReduceMatrixMultiplication:95  - Awaiting completion...
2022-03-16 11:57:53,784 INFO  [main] o.d.s.s.c.a.ParallelReduceMatrixMultiplication:100 - All tasks completed.
2022-03-16 11:57:53,785 INFO  [main] o.d.s.s.c.u.Chronometer                   :86  - Stopping chronometer...
2022-03-16 11:57:53,785 INFO  [main] o.d.s.s.c.u.Chronometer                   :97  - Chronometer stopped.
2022-03-16 11:57:53,785 WARN  [main] o.d.s.s.c.a.ParallelReduceMatrixMultiplication:74  - Duration: 977.699 ms
2022-03-16 11:57:53,785 INFO  [main] o.d.s.s.c.a.MultiplicationTestByMatrix    :92  - Validating result...
2022-03-16 11:57:53,786 INFO  [main] o.d.s.s.c.m.MatrixLoader                  :48  - Loading new matrix [largeC]...
2022-03-16 11:57:53,786 INFO  [main] o.d.s.s.c.m.Matrix                        :47  - Creating empty matrix...
2022-03-16 11:57:53,786 INFO  [main] o.d.s.s.c.m.MatrixLoader                  :69  - Loading rows from file [largeC.mtx]...
2022-03-16 11:57:53,788 INFO  [main] o.d.s.s.c.m.MatrixLoader                  :79  - Rows loaded.
2022-03-16 11:57:53,788 INFO  [main] o.d.s.s.c.m.MatrixLoader                  :95  - Estimating memory need...
2022-03-16 11:57:53,789 INFO  [main] o.d.s.s.c.m.MatrixLoader                  :101 - Memory needs: [200,100]
2022-03-16 11:57:53,789 INFO  [main] o.d.s.s.c.m.Matrix                        :56  - Allocating memory for [200,100] matrix...
2022-03-16 11:57:53,789 INFO  [main] o.d.s.s.c.m.Matrix                        :64  - Memory allocated.
2022-03-16 11:57:53,789 INFO  [main] o.d.s.s.c.m.Matrix                        :83  - Filling [200,100] matrix...
2022-03-16 11:57:53,789 INFO  [main] o.d.s.s.c.m.Matrix                        :84  - Copying matrix's file content...
2022-03-16 11:57:53,790 INFO  [main] o.d.s.s.c.m.Matrix                        :91  - Matrix's file content copied.
2022-03-16 11:57:53,790 INFO  [main] o.d.s.s.c.m.Matrix                        :140 - Comparing matrix [largeAxlargeB] with matrix [largeC]
2022-03-16 11:57:53,790 INFO  [main] o.d.s.s.c.m.Matrix                        :153 - Matrices match.
```

C Logs

Listing 26: error.log

```
16:20:36 [INFO ] LOG_UTIL   : Log file created.
16:20:36 [INFO ] LOG_UTIL   : Logger enabled.
16:20:36 [INFO ] LOG_UTIL   : Log level set to DEBUG
16:20:36 [INFO ] MAIN       : Performing Matrices Multiplication Test...
16:20:36 [INFO ] MAIN       : Selected matrices group: error
16:20:36 [INFO ] MAIN       : Setting-up...
16:20:36 [INFO ] MTX_LOADER : Loading new matrix [errorA]...
16:20:36 [INFO ] MATRIX     : Creating empty matrix...
16:20:36 [INFO ] MTX_LOADER : Estimating memory needs for file [errorA.mtx]...
16:20:36 [INFO ] FILE_UTIL  : Opening file [errorA.mtx] in r mode
16:20:36 [DEBUG] MTX_LOADER : Measuring line 0...
16:20:36 [DEBUG] MTX_LOADER : Width of line 0 = 3
16:20:36 [DEBUG] MTX_LOADER : Matrix's width updated from 0 to 3.
16:20:36 [DEBUG] MTX_LOADER : Measuring line 1...
16:20:36 [DEBUG] MTX_LOADER : Width of line 1 = 3
16:20:36 [DEBUG] MTX_LOADER : Measuring line 2...
16:20:36 [DEBUG] MTX_LOADER : Width of line 2 = 3
16:20:36 [INFO ] FILE_UTIL  : Closing file [errorA.mtx]
16:20:36 [INFO ] MTX_LOADER : Estimated needs: [3,3].
16:20:36 [INFO ] MATRIX     : Allocating memory for [3,3] matrix...
16:20:36 [INFO ] MATRIX     : Memory allocated.
16:20:36 [INFO ] MTX_LOADER : Filling [3,3] matrix from file [errorA.mtx]...
16:20:36 [INFO ] FILE_UTIL  : Opening file [errorA.mtx] in r mode
16:20:36 [INFO ] MTX_LOADER : Copying matrix's file content...
16:20:36 [DEBUG] MTX_LOADER : Processing lines...
16:20:36 [DEBUG] MTX_LOADER : Processing columns for line 0...
16:20:36 [DEBUG] MTX_LOADER : 3 columns processed.
16:20:36 [DEBUG] MTX_LOADER : Processing columns for line 1...
16:20:36 [DEBUG] MTX_LOADER : 3 columns processed.
16:20:36 [DEBUG] MTX_LOADER : Processing columns for line 2...
16:20:36 [DEBUG] MTX_LOADER : 3 columns processed.
16:20:36 [DEBUG] MTX_LOADER : 3 lines processed.
16:20:36 [INFO ] MTX_LOADER : Matrix's file content copied.
16:20:36 [INFO ] FILE_UTIL  : Closing file [errorA.mtx]
16:20:36 [INFO ] MTX_LOADER : Matrix filled.
16:20:36 [DEBUG] MATRIX     : Displaying matrix...
  1       0       0
  0       1       0
  0       0       1
16:20:36 [INFO ] MTX_LOADER : Matrix loaded
16:20:36 [INFO ] MTX_LOADER : Loading new matrix [errorB]...
16:20:36 [INFO ] MATRIX     : Creating empty matrix...
16:20:36 [INFO ] MTX_LOADER : Estimating memory needs for file [errorB.mtx]...
16:20:36 [INFO ] FILE_UTIL  : Opening file [errorB.mtx] in r mode
16:20:36 [DEBUG] MTX_LOADER : Measuring line 0...
16:20:36 [DEBUG] MTX_LOADER : Width of line 0 = 2
16:20:36 [DEBUG] MTX_LOADER : Matrix's width updated from 0 to 2.
16:20:36 [DEBUG] MTX_LOADER : Measuring line 1...
16:20:36 [DEBUG] MTX_LOADER : Width of line 1 = 2
16:20:36 [DEBUG] MTX_LOADER : Measuring line 2...
16:20:36 [DEBUG] MTX_LOADER : Width of line 2 = 2
16:20:36 [INFO ] FILE_UTIL  : Closing file [errorB.mtx]
16:20:36 [INFO ] MTX_LOADER : Estimated needs: [3,2].
16:20:36 [INFO ] MATRIX     : Allocating memory for [3,2] matrix...
16:20:36 [INFO ] MATRIX     : Memory allocated.
16:20:36 [INFO ] MTX_LOADER : Filling [3,2] matrix from file [errorB.mtx]...
16:20:36 [INFO ] FILE_UTIL  : Opening file [errorB.mtx] in r mode
16:20:36 [INFO ] MTX_LOADER : Copying matrix's file content...
16:20:36 [DEBUG] MTX_LOADER : Processing lines...
16:20:36 [DEBUG] MTX_LOADER : Processing columns for line 0...
16:20:36 [DEBUG] MTX_LOADER : 2 columns processed.
16:20:36 [DEBUG] MTX_LOADER : Processing columns for line 1...
16:20:36 [DEBUG] MTX_LOADER : 2 columns processed.
16:20:36 [DEBUG] MTX_LOADER : Processing columns for line 2...
16:20:36 [DEBUG] MTX_LOADER : 2 columns processed.
16:20:36 [DEBUG] MTX_LOADER : 3 lines processed.
16:20:36 [INFO ] MTX_LOADER : Matrix's file content copied.
16:20:36 [INFO ] FILE_UTIL  : Closing file [errorB.mtx]
16:20:36 [INFO ] MTX_LOADER : Matrix filled.
16:20:36 [DEBUG] MATRIX     : Displaying matrix...
  1       1
  2       2
  3       3
16:20:36 [INFO ] MTX_LOADER : Matrix loaded
16:20:36 [INFO ] MTX_LOADER : Loading new matrix [errorC]...
16:20:36 [INFO ] MATRIX     : Creating empty matrix...
16:20:36 [INFO ] MTX_LOADER : Estimating memory needs for file [errorC.mtx]...
16:20:36 [INFO ] FILE_UTIL  : Opening file [errorC.mtx] in r mode
16:20:36 [DEBUG] MTX_LOADER : Measuring line 0...
16:20:36 [DEBUG] MTX_LOADER : Width of line 0 = 2
16:20:36 [DEBUG] MTX_LOADER : Matrix's width updated from 0 to 2.
16:20:36 [DEBUG] MTX_LOADER : Measuring line 1...
16:20:36 [DEBUG] MTX_LOADER : Width of line 1 = 2
16:20:36 [DEBUG] MTX_LOADER : Measuring line 2...
16:20:36 [DEBUG] MTX_LOADER : Width of line 2 = 2
16:20:36 [INFO ] FILE_UTIL  : Closing file [errorC.mtx]
16:20:36 [INFO ] MTX_LOADER : Estimated needs: [3,2].
16:20:36 [INFO ] MATRIX     : Allocating memory for [3,2] matrix...
16:20:36 [INFO ] MATRIX     : Memory allocated.
16:20:36 [INFO ] MTX_LOADER : Filling [3,2] matrix from file [errorC.mtx]...
16:20:36 [INFO ] FILE_UTIL  : Opening file [errorC.mtx] in r mode
16:20:36 [INFO ] MTX_LOADER : Copying matrix's file content...
16:20:36 [DEBUG] MTX_LOADER : Processing lines...
16:20:36 [DEBUG] MTX_LOADER : Processing columns for line 0...
16:20:36 [DEBUG] MTX_LOADER : 2 columns processed.
16:20:36 [DEBUG] MTX_LOADER : Processing columns for line 1...
16:20:36 [DEBUG] MTX_LOADER : 2 columns processed.
16:20:36 [DEBUG] MTX_LOADER : Processing columns for line 2...
16:20:36 [DEBUG] MTX_LOADER : 2 columns processed.
16:20:36 [DEBUG] MTX_LOADER : 3 lines processed.
16:20:36 [INFO ] MTX_LOADER : Matrix's file content copied.
16:20:36 [INFO ] FILE_UTIL  : Closing file [errorC.mtx]
16:20:36 [INFO ] MTX_LOADER : Matrix filled.
16:20:36 [DEBUG] MATRIX     : Displaying matrix...
  0       0
  0       0
  0       0
16:20:36 [INFO ] MTX_LOADER : Matrix loaded
16:20:36 [INFO ] MAIN       : Computing product of errorA and errorB...
16:20:36 [INFO ] MAIN       : Using sequential method...
16:20:36 [INFO ] CHRONO_UTIL: Creating new chronometer...
16:20:36 [INFO ] CHRONO_UTIL: Chronometer created.
16:20:36 [INFO ] CHRONO_UTIL: Starting chronometer...
16:20:36 [INFO ] CHRONO_UTIL: Chronometer started.
16:20:36 [INFO ] MATRIX     : Creating new [3,2] matrix...
16:20:36 [INFO ] MATRIX     : Creating empty matrix...
16:20:36 [INFO ] MATRIX     : Allocating memory for [3,2] matrix...
16:20:36 [INFO ] MATRIX     : Memory allocated.
16:20:36 [INFO ] CHRONO_UTIL: Stopping chronometer...
16:20:36 [INFO ] CHRONO_UTIL: Chronometer stopped.
16:20:36 [INFO ] CHRONO_UTIL: Duration: 0.010 ms.
16:20:36 [DEBUG] MATRIX     : Displaying matrix...
  1       1
  2       2
  3       3
16:20:36 [INFO ] MTX_WRITER : Writing matrix errorAxerrorB to file...
16:20:36 [INFO ] FILE_UTIL  : Opening file [errorAxerrorB.mtx] in wb mode
16:20:36 [INFO ] FILE_UTIL  : Closing file [errorAxerrorB.mtx]
16:20:36 [INFO ] MTX_WRITER : Matrix errorAxerrorB written to file.
16:20:36 [INFO ] MAIN       : Asserting computation is correct...
16:20:36 [INFO ] MATRIX     : Comparing matrix [errorC] with matrix [errorAxerrorB]
16:20:36 [INFO ] MATRIX     : On [0,0], matrix [errorC] (0) does not match with
       matrix [errorAxerrorB] (1)
16:20:36 [FAIL ] MAIN       : Exception occurred in MAIN: 6-ASSERTION_ERROR
Exception message: Product of errorA and errorB does not match errorC
```

Listing 27: missing.log

```
16:20:36 [INFO ] LOG_UTIL   : Log file created.
16:20:36 [INFO ] LOG_UTIL   : Logger enabled.
16:20:36 [INFO ] LOG_UTIL   : Log level set to DEBUG
16:20:36 [INFO ] MAIN       : Performing Matrices Multiplication Test...
16:20:36 [INFO ] MAIN       : Selected matrices group: missing
16:20:36 [INFO ] MAIN       : Setting-up...
16:20:36 [INFO ] MTX_LOADER : Loading new matrix [missingA]...
16:20:36 [INFO ] MATRIX     : Creating empty matrix...
16:20:36 [INFO ] MTX_LOADER : Estimating memory needs for file [missingA.mtx]...
16:20:36 [INFO ] FILE_UTIL  : Opening file [missingA.mtx] in r mode
16:20:36 [FAIL ] FILE_UTIL  : Exception occurred in FILE_UTIL: 5-FILE_NOT_FOUND
Exception message: File [missingA.mtx] not found!
```

Listing 28: mismatch.log

```
16:20:36 [INFO ] LOG_UTIL   : Log file created.
16:20:36 [INFO ] LOG_UTIL   : Logger enabled.
16:20:36 [INFO ] LOG_UTIL   : Log level set to DEBUG
16:20:36 [INFO ] MAIN       : Performing Matrices Multiplication Test...
16:20:36 [INFO ] MAIN       : Selected matrices group: mismatch
16:20:36 [INFO ] MAIN       : Setting-up...
16:20:36 [INFO ] MTX_LOADER : Loading new matrix [mismatchA]...
16:20:36 [INFO ] MATRIX     : Creating empty matrix...
16:20:36 [INFO ] MTX_LOADER : Estimating memory needs for file [mismatchA.mtx]...
16:20:36 [INFO ] FILE_UTIL  : Opening file [mismatchA.mtx] in r mode
16:20:36 [DEBUG] MTX_LOADER : Measuring line 0...
16:20:36 [DEBUG] MTX_LOADER : Width of line 0 = 3
16:20:36 [DEBUG] MTX_LOADER : Matrix's width updated from 0 to 3.
16:20:36 [DEBUG] MTX_LOADER : Measuring line 1...
16:20:36 [DEBUG] MTX_LOADER : Width of line 1 = 3
16:20:36 [DEBUG] MTX_LOADER : Measuring line 2...
16:20:36 [DEBUG] MTX_LOADER : Width of line 2 = 3
16:20:36 [INFO ] FILE_UTIL  : Closing file [mismatchA.mtx]
16:20:36 [INFO ] MTX_LOADER : Estimated needs: [3,3].
16:20:36 [INFO ] MATRIX     : Allocating memory for [3,3] matrix...
16:20:36 [INFO ] MATRIX     : Memory allocated.
16:20:36 [INFO ] MTX_LOADER : Filling [3,3] matrix from file [mismatchA.mtx]...
16:20:36 [INFO ] FILE_UTIL  : Opening file [mismatchA.mtx] in r mode
16:20:36 [INFO ] MTX_LOADER : Copying matrix's file content...
16:20:36 [DEBUG] MTX_LOADER : Processing lines...
16:20:36 [DEBUG] MTX_LOADER : Processing columns for line 0...
16:20:36 [DEBUG] MTX_LOADER : 3 columns processed.
16:20:36 [DEBUG] MTX_LOADER : Processing columns for line 1...
16:20:36 [DEBUG] MTX_LOADER : 3 columns processed.
16:20:36 [DEBUG] MTX_LOADER : Processing columns for line 2...
16:20:36 [DEBUG] MTX_LOADER : 3 columns processed.
16:20:36 [DEBUG] MTX_LOADER : 3 lines processed.
16:20:36 [INFO ] MTX_LOADER : Matrix's file content copied.
16:20:36 [INFO ] FILE_UTIL  : Closing file [mismatchA.mtx]
16:20:36 [INFO ] MTX_LOADER : Matrix filled.
16:20:36 [DEBUG] MATRIX     : Displaying matrix...
  1       0       0
  0       1       0
  0       0       1
16:20:36 [INFO ] MTX_LOADER : Matrix loaded
16:20:36 [INFO ] MTX_LOADER : Loading new matrix [mismatchB]...
16:20:36 [INFO ] MATRIX     : Creating empty matrix...
16:20:36 [INFO ] MTX_LOADER : Estimating memory needs for file [mismatchB.mtx]...
16:20:36 [INFO ] FILE_UTIL  : Opening file [mismatchB.mtx] in r mode
16:20:36 [DEBUG] MTX_LOADER : Measuring line 0...
16:20:36 [DEBUG] MTX_LOADER : Width of line 0 = 2
16:20:36 [DEBUG] MTX_LOADER : Matrix's width updated from 0 to 2.
16:20:36 [DEBUG] MTX_LOADER : Measuring line 1...
16:20:36 [DEBUG] MTX_LOADER : Width of line 1 = 2
16:20:36 [DEBUG] MTX_LOADER : Measuring line 2...
16:20:36 [DEBUG] MTX_LOADER : Width of line 2 = 2
16:20:36 [DEBUG] MTX_LOADER : Measuring line 3...
16:20:36 [DEBUG] MTX_LOADER : Width of line 3 = 2
16:20:36 [INFO ] FILE_UTIL  : Closing file [mismatchB.mtx]
16:20:36 [INFO ] MTX_LOADER : Estimated needs: [4,2].
16:20:36 [INFO ] MATRIX     : Allocating memory for [4,2] matrix...
16:20:36 [INFO ] MATRIX     : Memory allocated.
16:20:36 [INFO ] MTX_LOADER : Filling [4,2] matrix from file [mismatchB.mtx]...
16:20:36 [INFO ] FILE_UTIL  : Opening file [mismatchB.mtx] in r mode
16:20:36 [INFO ] MTX_LOADER : Copying matrix's file content...
16:20:36 [DEBUG] MTX_LOADER : Processing lines...
16:20:36 [DEBUG] MTX_LOADER : Processing columns for line 0...
16:20:36 [DEBUG] MTX_LOADER : 2 columns processed.
16:20:36 [DEBUG] MTX_LOADER : Processing columns for line 1...
16:20:36 [DEBUG] MTX_LOADER : 2 columns processed.
16:20:36 [DEBUG] MTX_LOADER : Processing columns for line 2...
16:20:36 [DEBUG] MTX_LOADER : 2 columns processed.
16:20:36 [DEBUG] MTX_LOADER : Processing columns for line 3...
16:20:36 [DEBUG] MTX_LOADER : 2 columns processed.
16:20:36 [DEBUG] MTX_LOADER : 4 lines processed.
16:20:36 [INFO ] MTX_LOADER : Matrix's file content copied.
16:20:36 [INFO ] FILE_UTIL  : Closing file [mismatchB.mtx]
16:20:36 [INFO ] MTX_LOADER : Matrix filled.
16:20:36 [DEBUG] MATRIX     : Displaying matrix...
  1       1
  2       2
  3       3
  4       4
16:20:36 [INFO ] MTX_LOADER : Matrix loaded
16:20:36 [INFO ] MTX_LOADER : Loading new matrix [mismatchC]...
16:20:36 [INFO ] MATRIX     : Creating empty matrix...
16:20:36 [INFO ] MTX_LOADER : Estimating memory needs for file [mismatchC.mtx]...
16:20:36 [INFO ] FILE_UTIL  : Opening file [mismatchC.mtx] in r mode
16:20:36 [INFO ] FILE_UTIL  : Closing file [mismatchC.mtx]
16:20:36 [INFO ] MTX_LOADER : Estimated needs: [0,0].
16:20:36 [INFO ] MATRIX     : Allocating memory for [0,0] matrix...
16:20:36 [INFO ] MATRIX     : Memory allocated.
16:20:36 [INFO ] MTX_LOADER : Filling [0,0] matrix from file [mismatchC.mtx]...
16:20:36 [INFO ] FILE_UTIL  : Opening file [mismatchC.mtx] in r mode
16:20:36 [INFO ] MTX_LOADER : Copying matrix's file content...
16:20:36 [DEBUG] MTX_LOADER : Processing lines...
16:20:36 [DEBUG] MTX_LOADER : 0 lines processed.
16:20:36 [INFO ] MTX_LOADER : Matrix's file content copied.
16:20:36 [INFO ] FILE_UTIL  : Closing file [mismatchC.mtx]
16:20:36 [INFO ] MTX_LOADER : Matrix filled.
16:20:36 [DEBUG] MATRIX     : Displaying matrix...
16:20:36 [INFO ] MTX_LOADER : Matrix loaded
16:20:36 [INFO ] MAIN       : Computing product of mismatchA and mismatchB...
16:20:36 [INFO ] MAIN       : Using sequential method...
16:20:36 [FAIL ] MTX_MULTI  : Exception occurred in MTX_MULTI:
       2-INVALID_MATRIX_DIMENSION
Exception message: Matrix mismatchA [3,3] cannot be multiplied with matrix mismatchB
       [4,2]
```

Listing 29: large–sequential.log

```
16:20:36 [INFO ] LOG_UTIL   : Log file created.
16:20:36 [INFO ] LOG_UTIL   : Logger enabled.
16:20:36 [INFO ] MAIN       : Performing Matrices Multiplication Test...
16:20:36 [INFO ] MAIN       : Selected matrices group: large
16:20:36 [INFO ] MAIN       : Setting-up...
16:20:36 [INFO ] MTX_LOADER : Loading new matrix [largeA]...
16:20:36 [INFO ] MATRIX     : Creating empty matrix...
16:20:36 [INFO ] MTX_LOADER : Estimating memory needs for file [largeA.mtx]...
16:20:36 [INFO ] FILE_UTIL  : Opening file [largeA.mtx] in r mode
16:20:36 [INFO ] FILE_UTIL  : Closing file [largeA.mtx]
16:20:36 [INFO ] MTX_LOADER : Estimated needs: [10,12].
16:20:36 [INFO ] MATRIX     : Allocating memory for [10,12] matrix...
16:20:36 [INFO ] MATRIX     : Memory allocated.
16:20:36 [INFO ] MTX_LOADER : Filling [10,12] matrix from file [largeA.mtx]...
16:20:36 [INFO ] FILE_UTIL  : Opening file [largeA.mtx] in r mode
16:20:36 [INFO ] MTX_LOADER : Copying matrix's file content...
16:20:36 [INFO ] MTX_LOADER : Matrix's file content copied.
16:20:36 [INFO ] FILE_UTIL  : Closing file [largeA.mtx]
16:20:36 [INFO ] MTX_LOADER : Matrix filled.
16:20:36 [INFO ] MTX_LOADER : Matrix loaded
16:20:36 [INFO ] MTX_LOADER : Loading new matrix [largeB]...
16:20:36 [INFO ] MATRIX     : Creating empty matrix...
16:20:36 [INFO ] MTX_LOADER : Estimating memory needs for file [largeB.mtx]...
16:20:36 [INFO ] FILE_UTIL  : Opening file [largeB.mtx] in r mode
16:20:36 [INFO ] FILE_UTIL  : Closing file [largeB.mtx]
16:20:36 [INFO ] MTX_LOADER : Estimated needs: [12,3].
16:20:36 [INFO ] MATRIX     : Allocating memory for [12,3] matrix...
16:20:36 [INFO ] MATRIX     : Memory allocated.
16:20:36 [INFO ] MTX_LOADER : Filling [12,3] matrix from file [largeB.mtx]...
16:20:36 [INFO ] FILE_UTIL  : Opening file [largeB.mtx] in r mode
16:20:36 [INFO ] MTX_LOADER : Copying matrix's file content...
16:20:36 [INFO ] MTX_LOADER : Matrix's file content copied.
16:20:36 [INFO ] FILE_UTIL  : Closing file [largeB.mtx]
16:20:36 [INFO ] MTX_LOADER : Matrix filled.
16:20:36 [INFO ] MTX_LOADER : Matrix loaded
16:20:36 [INFO ] MTX_LOADER : Loading new matrix [largeC]...
16:20:36 [INFO ] MATRIX     : Creating empty matrix...
16:20:36 [INFO ] MTX_LOADER : Estimating memory needs for file [largeC.mtx]...
16:20:36 [INFO ] FILE_UTIL  : Opening file [largeC.mtx] in r mode
16:20:36 [INFO ] FILE_UTIL  : Closing file [largeC.mtx]
16:20:36 [INFO ] MTX_LOADER : Estimated needs: [10,3].
16:20:36 [INFO ] MATRIX     : Allocating memory for [10,3] matrix...
16:20:36 [INFO ] MATRIX     : Memory allocated.
16:20:36 [INFO ] MTX_LOADER : Filling [10,3] matrix from file [largeC.mtx]...
16:20:36 [INFO ] FILE_UTIL  : Opening file [largeC.mtx] in r mode
16:20:36 [INFO ] MTX_LOADER : Copying matrix's file content...
16:20:36 [INFO ] MTX_LOADER : Matrix's file content copied.
16:20:36 [INFO ] FILE_UTIL  : Closing file [largeC.mtx]
16:20:36 [INFO ] MTX_LOADER : Matrix filled.
16:20:36 [INFO ] MTX_LOADER : Matrix loaded
16:20:36 [INFO ] MAIN       : Computing product of largeA and largeB...
16:20:36 [INFO ] MAIN       : Using sequential method...
16:20:36 [INFO ] CHRONO_UTIL: Creating new chronometer...
16:20:36 [INFO ] CHRONO_UTIL: Chronometer created.
16:20:36 [INFO ] CHRONO_UTIL: Starting chronometer...
16:20:36 [INFO ] CHRONO_UTIL: Chronometer started.
16:20:36 [INFO ] MATRIX     : Creating new [10,3] matrix...
16:20:36 [INFO ] MATRIX     : Creating empty matrix...
16:20:36 [INFO ] MATRIX     : Allocating memory for [10,3] matrix...
16:20:36 [INFO ] MATRIX     : Memory allocated.
16:20:36 [INFO ] CHRONO_UTIL: Stopping chronometer...
16:20:36 [INFO ] CHRONO_UTIL: Chronometer stopped.
16:20:36 [INFO ] CHRONO_UTIL: Duration: 0.011 ms.
16:20:36 [INFO ] MTX_WRITER : Writing matrix largeAxlargeB to file...
16:20:36 [INFO ] FILE_UTIL  : Opening file [largeAxlargeB.mtx] in wb mode
16:20:36 [INFO ] FILE_UTIL  : Closing file [largeAxlargeB.mtx]
16:20:36 [INFO ] MTX_WRITER : Matrix largeAxlargeB written to file.
16:20:36 [INFO ] MAIN       : Asserting computation is correct...
16:20:36 [INFO ] MATRIX     : Comparing matrix [largeC] with matrix [largeAxlargeB]
16:20:36 [INFO ] MAIN       : Product of largeA and largeB is correct!
16:20:36 [INFO ] MATRIX     : Destroying matrix [largeAxlargeB]...
16:20:36 [INFO ] MAIN       : Cleaning-up...
16:20:36 [INFO ] MATRIX     : Destroying matrix [largeA]...
16:20:36 [INFO ] MATRIX     : Destroying matrix [largeB]...
16:20:36 [INFO ] MATRIX     : Destroying matrix [largeC]...
16:20:36 [INFO ] MAIN       : Test completed
16:20:36 [INFO ] LOG_UTIL   : Logger disabled.
```

Listing 30: large–parallel.log

```
16:20:36 [INFO ] LOG_UTIL   : Log file created.
16:20:36 [INFO ] LOG_UTIL   : Logger enabled.
16:20:36 [INFO ] MAIN       : Performing Matrices Multiplication Test...
16:20:36 [INFO ] MAIN       : Selected matrices group: large
16:20:36 [INFO ] MAIN       : Setting-up...
16:20:36 [INFO ] MTX_LOADER : Loading new matrix [largeA]...
16:20:36 [INFO ] MATRIX     : Creating empty matrix...
16:20:36 [INFO ] MTX_LOADER : Estimating memory needs for file [largeA.mtx]...
16:20:36 [INFO ] FILE_UTIL  : Opening file [largeA.mtx] in r mode
16:20:36 [INFO ] FILE_UTIL  : Closing file [largeA.mtx]
16:20:36 [INFO ] MTX_LOADER : Estimated needs: [10,12].
16:20:36 [INFO ] MATRIX     : Allocating memory for [10,12] matrix...
16:20:36 [INFO ] MATRIX     : Memory allocated.
16:20:36 [INFO ] MTX_LOADER : Filling [10,12] matrix from file [largeA.mtx]...
16:20:36 [INFO ] FILE_UTIL  : Opening file [largeA.mtx] in r mode
16:20:36 [INFO ] MTX_LOADER : Copying matrix's file content...
16:20:36 [INFO ] MTX_LOADER : Matrix's file content copied.
16:20:36 [INFO ] FILE_UTIL  : Closing file [largeA.mtx]
16:20:36 [INFO ] MTX_LOADER : Matrix filled.
16:20:36 [INFO ] MTX_LOADER : Matrix loaded
16:20:36 [INFO ] MTX_LOADER : Loading new matrix [largeB]...
16:20:36 [INFO ] MATRIX     : Creating empty matrix...
16:20:36 [INFO ] MTX_LOADER : Estimating memory needs for file [largeB.mtx]...
16:20:36 [INFO ] FILE_UTIL  : Opening file [largeB.mtx] in r mode
16:20:36 [INFO ] FILE_UTIL  : Closing file [largeB.mtx]
16:20:36 [INFO ] MTX_LOADER : Estimated needs: [12,3].
16:20:36 [INFO ] MATRIX     : Allocating memory for [12,3] matrix...
16:20:36 [INFO ] MATRIX     : Memory allocated.
16:20:36 [INFO ] MTX_LOADER : Filling [12,3] matrix from file [largeB.mtx]...
16:20:36 [INFO ] FILE_UTIL  : Opening file [largeB.mtx] in r mode
16:20:36 [INFO ] MTX_LOADER : Copying matrix's file content...
16:20:36 [INFO ] MTX_LOADER : Matrix's file content copied.
16:20:36 [INFO ] FILE_UTIL  : Closing file [largeB.mtx]
16:20:36 [INFO ] MTX_LOADER : Matrix filled.
16:20:36 [INFO ] MTX_LOADER : Matrix loaded
16:20:36 [INFO ] MTX_LOADER : Loading new matrix [largeC]...
16:20:36 [INFO ] MATRIX     : Creating empty matrix...
16:20:36 [INFO ] MTX_LOADER : Estimating memory needs for file [largeC.mtx]...
16:20:36 [INFO ] FILE_UTIL  : Opening file [largeC.mtx] in r mode
16:20:36 [INFO ] FILE_UTIL  : Closing file [largeC.mtx]
16:20:36 [INFO ] MTX_LOADER : Estimated needs: [10,3].
16:20:36 [INFO ] MATRIX     : Allocating memory for [10,3] matrix...
16:20:36 [INFO ] MATRIX     : Memory allocated.
16:20:36 [INFO ] MTX_LOADER : Filling [10,3] matrix from file [largeC.mtx]...
16:20:36 [INFO ] FILE_UTIL  : Opening file [largeC.mtx] in r mode
16:20:36 [INFO ] MTX_LOADER : Copying matrix's file content...
16:20:36 [INFO ] MTX_LOADER : Matrix's file content copied.
16:20:36 [INFO ] FILE_UTIL  : Closing file [largeC.mtx]
16:20:36 [INFO ] MTX_LOADER : Matrix filled.
16:20:36 [INFO ] MTX_LOADER : Matrix loaded
16:20:36 [INFO ] MAIN       : Computing product of largeA and largeB...
16:20:36 [INFO ] MAIN       : Using parallel method...
16:20:36 [INFO ] CHRONO_UTIL: Creating new chronometer...
16:20:36 [INFO ] CHRONO_UTIL: Chronometer created.
16:20:36 [INFO ] CHRONO_UTIL: Starting chronometer...
16:20:36 [INFO ] CHRONO_UTIL: Chronometer started.
16:20:36 [INFO ] MATRIX     : Creating new [10,3] matrix...
16:20:36 [INFO ] MATRIX     : Creating empty matrix...
16:20:36 [INFO ] MATRIX     : Allocating memory for [10,3] matrix...
16:20:36 [INFO ] MATRIX     : Memory allocated.
16:20:36 [INFO ] CHRONO_UTIL: Stopping chronometer...
16:20:36 [INFO ] CHRONO_UTIL: Chronometer stopped.
16:20:36 [INFO ] CHRONO_UTIL: Duration: 0.193 ms.
16:20:36 [INFO ] MTX_WRITER : Writing matrix largeAxlargeB to file...
16:20:36 [INFO ] FILE_UTIL  : Opening file [largeAxlargeB.mtx] in wb mode
16:20:36 [INFO ] FILE_UTIL  : Closing file [largeAxlargeB.mtx]
16:20:36 [INFO ] MTX_WRITER : Matrix largeAxlargeB written to file.
16:20:36 [INFO ] MAIN       : Asserting computation is correct...
16:20:36 [INFO ] MATRIX     : Comparing matrix [largeC] with matrix [largeAxlargeB]
16:20:36 [INFO ] MAIN       : Product of largeA and largeB is correct!
16:20:36 [INFO ] MATRIX     : Destroying matrix [largeAxlargeB]...
16:20:36 [INFO ] MAIN       : Cleaning-up...
16:20:36 [INFO ] MATRIX     : Destroying matrix [largeA]...
16:20:36 [INFO ] MATRIX     : Destroying matrix [largeB]...
16:20:36 [INFO ] MATRIX     : Destroying matrix [largeC]...
16:20:36 [INFO ] MAIN       : Test completed
16:20:36 [INFO ] LOG_UTIL   : Logger disabled.
```