Lab 9a: Matrix Multiplication

- Deadline: 15 November, 2022, Tuesday, 23:59, SST
- Mark: 2%

Files

You are given the following files:

- `Lab9a.java`: The main function.
- `Matrix.java`: A matrix containing the following useful methods:
  - `nonRecursiveMultiply`: The typical matrix multiplication algorithm
  - `recursiveMultiply`: A recursive but non-parallel matrix multiplication algorithm
- `MatrixMultiplication.java`: The template for your solution

There are test cases inside `input` and `output` folder. You are not allowed to change `Lab9a.java` and `Matrix.java`. Any modification will nullify your mark immediately.

Problem Description

Matrix multiplication is a fundamental operation with many applications in physics, engineering, mathematics, and computer science.

Given a matrix $A_{n \times m}$ of n rows by m columns, and a matrix $B_{m \times p}$, the matrix product $C_{n \times p}$ = AB is an has elements c_{ij} given by

$c_{ij} = \sum^{m}_{k=1} m_k = a_{ik} b_{kj}$

We are interested in parallelizing the following divide-and-conquer algorithm for matrix multiplication. Let:

```
A = [ A_11 | A_12 ]
    [ A_21 | A_22 ]

B = [ B_11 | B_12 ]
    [ B_21 | B_22 ]

C = [ C_11 | C_12 ]
    [ C_21 | C_22 ]
```

where A_{11}, A_{12}, *etc.* are block partitioned matrices of **equal** sizes. If C = AB, then:

```
C = [ C_11 | C_12 ]
    [ C_21 | C_22 ]

  = [ A_11 | A_12 ] [ B_11 | B_12 ]
    [ A_21 | A_22 ] [ B_21 | B_22 ]

  = [ A_11 B_11 + A_12 B_21  |  A_11 B_12 + A_12 B_22 ]
    [ A_21 B_11 + A_22 B_21  |  A_21 B_12 + A_22 B_22 ]
```

Note that A_{11} B_{11} is a matrix multiplication. In particular, (A_{11} B_{11}) + (A_{12} B_{21}) is a matrix multiplication followed by matrix addition.

You may want to study the `recursiveMultiply` to understand this algorithm better.

The Task

You are to implement the above divide-and-conquer algorithm as a `RecursiveTask` and submit it to `ForkJoinPool` for execution. For simplicity, we only need to handle square matrices of size 2^n for n up to 11. For this large number, the execution will not be on CodeCrunch but will be run on `stu.comp.nus.edu.sg`. You do not need to do anything for this large input as it will be based on your CodeCrunch submission.

A skeleton file `MatrixMultiplication.java` has been provided for you. The class `MatrixMultiplication` inherits from `RecursiveTask<Matrix>`, with the necessary fields and constructor. Your task is to complete the `compute` method.

The file `Matrix.java` is also provided for you. It implements a matrix with `double` values, and stores the values of the matrix in a 2D `double` array called `m`. It also stores the dimensions of the matrix in the field `dimension`. It includes two methods to multiply two matrices, one sequentially with triple for loops, and another (also sequentially) with the recursive divide-and-conquer algorithms. There is a method to compare if two matrices are equal.

In addition, the method `parallelMultiply` invokes the parallel version of matrix multiplication. At this moment, the method simply calls the non-parallel version of `recursiveMultiply`. You are to modify the method to implement the parallel version of recursive matrix multiplication.

The driver class called `Lab9a.java` is also provided for you which reads the input matrix, call the `parallelMultiply` and prints the resulting matrix. Due to the use of `double`, we only care about precision up to 3 decimal places.

Points to note:

- Find a suitable `FORK_THRESHOLD` for `MatrixMultiplication` such that any matrix dimension smaller than this threshold would be better off using sequential matrix multiplication.
- Try with small matrices first. Make sure the code is correct before you go for larger matrices.
- You should not spawn too many tasks that block, which will in turn lead to too many compensation threads being created in `ForkJoinPool`, and a `RejectedExecutionException` being thrown.
- You should not let multiple tasks update the same matrix in place. Such side effects may lead to incorrect results. For matrices of dimensions 2^{10} and 2^{11}, you need to run java with the argument `-Xmx[size]` to increase the heap memory size. For example, `-Xmx1g` increases the heap memory up to 1GB, and should work well for both cases. That said, you should still not create too many unnecessary copies of the matrices.
  - You do not have to worry too much about this unless you want to ensure that your code can achieve a good speedup.
  - The test on CodeCrunch only goes up to 2^9 due to the memory limitation on CodeCrunch.
- If you grow impatient while waiting and want to stop the running process, type Control-C in your ssh window. You may have to wait up to a few seconds for the process to stop.

Submission

Submit only the following files:

- `Lab9a.java`
- `Matrix.java`
- `MatrixMultiplication.java`