A simple thread simulator for parallelism

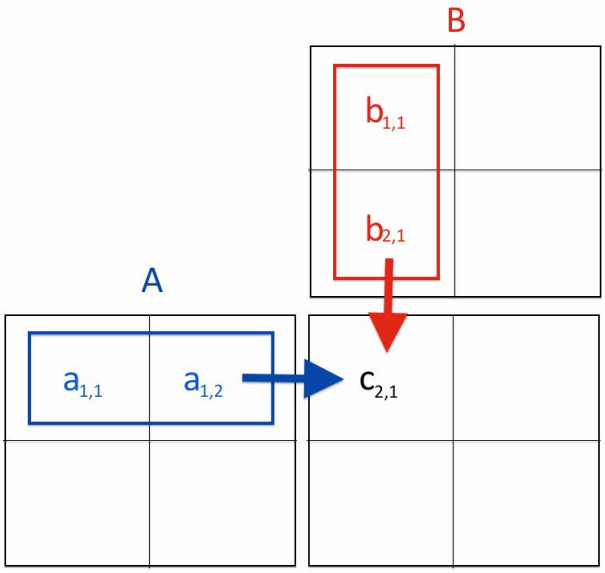
For this project, you will use Java thread to implement a matrix production operation with “columnGeneration” as the main task of a thread.

Background

Before starting, it is helpful to briefly recap how a matrix-matrix multiplication is computed. Let's say we have two matrices, A and B. Assume that A is a n×m matrix, which means that it has n rows and m columns. Also assume that B is a m×w matrix. The result of the multiplication A∗B (which is different from B∗A!) is a n×w matrix, which we call C. That is, the number of rows in the resulting matrix equals the number of rows of the first matrix A and the number of columns of the second matrix B.

Why does this happen and how does it work? The answer is the same for both questions here. Let's take the cell 1, 1 (first row, first column) of M. The number inside it after the operation C=A∗B is the sum of all the element-wise multiplications of the numbers in A, row 1, with the numbers in B, column 1. That is, in the cell i, j of C we have the sum of the element-wise multiplication of all the numbers in the i-th row in A and the j-th column in B.

The following figure intuitively explains this idea:



It should be pretty clear now why matrix-matrix multiplication is a good example for parallel computation. We have to compute every element in C, and each of them is independent from the others, so we can efficiently parallelise. Especially for modern

video or graphics cards processing, matrix multiplication plays a very important role for speed up rendering in video games.

Project Goal

The goal of this project is to use Java threads for matrix multiplication. As demonstrated in the class, we create a Matrix for matrix multiplication. However, it only used loops without parallel computation. If the two matrices have big sizes, e.g. 1000x1000, the multiplication between them could take a minute to get a result. So your job is to use thread to speed-up the matrix multiplication.

Implementation Details

All the places marked with "//Implementation here..." in the given code "Matrix.java" should be completed. Below are the details of each part.

(1) Finish the second constructor of Matrix class "Matrix(int r, int c, double v[][])"

Create the matrix with "r" rows and "c" columns and assign the elements with the third argument "v[]", which is a double 2D array. You need to do some checking to make sure the input 2D array should have the rows and columns with the same values as or greater values than the first two input arguments. It is fine if the input 2D array contains more numbers than the specified "r" rows and "c" columns needed. You can simply ignore those extra numbers. Similarly, if the input 2D array contains less numbers than "r" and "c", you can set the remaining elements with "0". (Hint: to check the size of the array, use the "length" data member of an array object)

(2) Finish the function "Matrix multiplyBy(Matrix m)"

This function is mostly implemented as demonstrated in the class. But you need to perform a dimension checking first to make sure the two matrices can do the multiplication. If not, the function just simply prints out a message and returns a null matrix.

(3) Finish the function "Matrix multiplyByThreads(Matrix m)"

This function multiplies the second matrix m and returns the result matrix by using threads. The details are provided in the comments of the provided code. To create multiple threads for each column, you need to use the Java Thread that triggers the run() function that is defined in the ColumnCalculator class below. To avoid over-creating threads, you can allocate 10 threads at a time and wait all the 10 threads to complete then initiate the next 10 threads. Also, please refer the "thread.join()" function about threads waiting scheme.

(4) Finish the function “void run()” defined in the ColumnCalculator class
The “run()” function is required for any instance that implements the Java Runnable interface. The job of run() function here is to compute one column (specified by the “col_idx” index) of the result matrix that is calculated by the sum-of-pairwise-product between the corresponding row of m1 and the corresponding column of m2.

(5) Finish the “main()” function to test your result
First, you can call the constructor you defined in (1) to create two simple matrices with easy numbers. Then you test if the “multiplyByThreads()” gives the correct result. Second you create two big matrices, e.g. 1000x2000, and apply both “multiplyBy()” and “multiplyByThreads()” functions to do the product between them and measure the time cost by using each function and write a simple report on a Word document or “.txt” file about their performances.

Rubrics:

1. Implementation (1) 15%
2. Implementation (2) 10%
3. Implementation (3) 35%
4. Implementation (4) 20%
5. Implementation (5) 20%

Submission:
For this project, you need to turn in both the “Matrix.java” code and a simple report (required in Implementation (5)) on the multiplication contrast between with and without threads. So zip both files and submit it to the Isidore.