Lab 9: Parallel Matrix Multiplication with ForkJoinPool

Submission deadline: 2359, Sunday, November 5, 2017.

Setup

The skeleton code from Lab 9 is available on `cs2030-i` under the directory `~cs2030/lab09`. `Matrix.java` is the main file that you need to edit to parallelize the matrix multiplication inside.

Background: Matrix Multiplication

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

Given a matrix A of $n \times m$ (n rows, m columns), and a matrix B of $m \times p$, the matrix produce $C = AB$ is an $n \times p$ matrix, where elements c_{ij} in C is given by: $c_{ij} = \sum_{k=1}^{m} a_{ik}b_{kj}$.

Task

You are given a class `Matrix` that implements a matrix with double values, and a method to multiply two matrices together -- the method `multiply` performs the multiplication using a straight forward method with triple `for` loops.

Your task is to parallelize `multiply`, by implementing `multiplyInParallel`.

You are still required to

- follows the CS2030 Coding Style [../style/index.html]
- clearly documented with `javadoc` [../javadoc/index.html]

Provided Classes

The class `Matrix` is given. It stores the values of the matrix in a 2D array `m` . It stores the number of rows `h` and number of columns `w` . The code is self-explanatory.

Parallelizing Matrix Multiplication

Matrix multiplication is an embarrassingly parallel operation. Each output value in the resulting matrix can be computed independently of other values. Furthermore, to compute the output value, we multiple two (potentially long) vectors together, another operation that can be easily parallelized.

You task is to write a parallel version of the matrix multiplication by creating `RecursiveTask` and submit them to the `ForkJoinPool` for execution using `fork` and `join` .

A challenge for this lab is to find the right level of parallelism that makes the code run faster rather than slower.

You can use any parallelization algorithm you like. Parallelizing calculation of each c_{ij} is a good place to start. You may also consider the more sophisticated divide and conquer algorithm [https://en.wikipedia.org/wiki/Matrix_multiplication#Parallel_matrix_multiplication], although it is more complicated and I can't seem to get more than 10 times speed up out of it.

Running on `sunfire`

You need to get the code and submit your solution on `cs2030-i` per usual. The VM `cs2030-i`, however, has only one processor and therefore it is not much fun to run parallel programs on it.

Fortunately, the host `sunfire`, has 256 processors[1] [#fn:1]. We will run your submitted solution on `sunfire`, and you can test your code on `sunfire` as well. You can `ssh` into `sunfire` just like `cs2030-i`.

Sample Results

I am able to get about 20 times speed up with the input:

```
java LabNine 200 25000 200
```

and about 10 times speed up with the input:

```
java LabNine 2000 2500 2000
```

Your solution should get similar or better speed up on `sunfire`.

Tips
Here are some tips:

- Try with small matrices first. Make sure the code is correct before you go for the larger matrices.
- You shouldn't run out of heap memory space for the huge matrices above. But if you algorithm requires more memory, you can run `java` with argument `-Xmx<size>` to increase the heap memory. For example, `java -Xmx256m LabNine 100 100 100` will run `LabNine` with heap size of 256 MB.
- Avoid copy the matrices unless absolutely necessary (copying large matrices incurs memory overhead).

Grading

This lab contributes another 4 marks to your final grade (100 marks).

You get:

- 1 mark if you parallelize the code but achieve a slow down in performance.
- 2 marks if you parallelize the code, manage to speed up, but is at least half as slow as my version (e.g., no more than 10 times / 5 times faster in the sample input above)
- 4 marks if you parallelize the code and manage to speed up as fast, or faster than my version.

You can get -0.5 mark deduction for serious violation of style. If you code is incorrect, you will get deduction depending on how serious your bug is.

Submission

When you are ready to submit your lab, on `cs2030-i` , run the script

```
~cs2030/submit09
```

which will copy all files matching `*.java` (and nothing else) from your `~/lab09` directory on `cs2030-i` to an internal grading directory. We will test compile and test run with a tiny sample input to make sure that your submission is OK.

You can submit multiple times, but only the most recent submission will be graded.

Warning
Make sure your code are in the right place -- it must be in subdirectory named `lab09` , directly under your home directory, in other words `~/lab09` . If you place it anywhere else, it will not get submitted.

---

1. You can find out by calling `Runtime.getRuntime().availableProcessors();` in Java.