Jacob Klingler
Operating Systems Lab 4
12 / 2 / 12

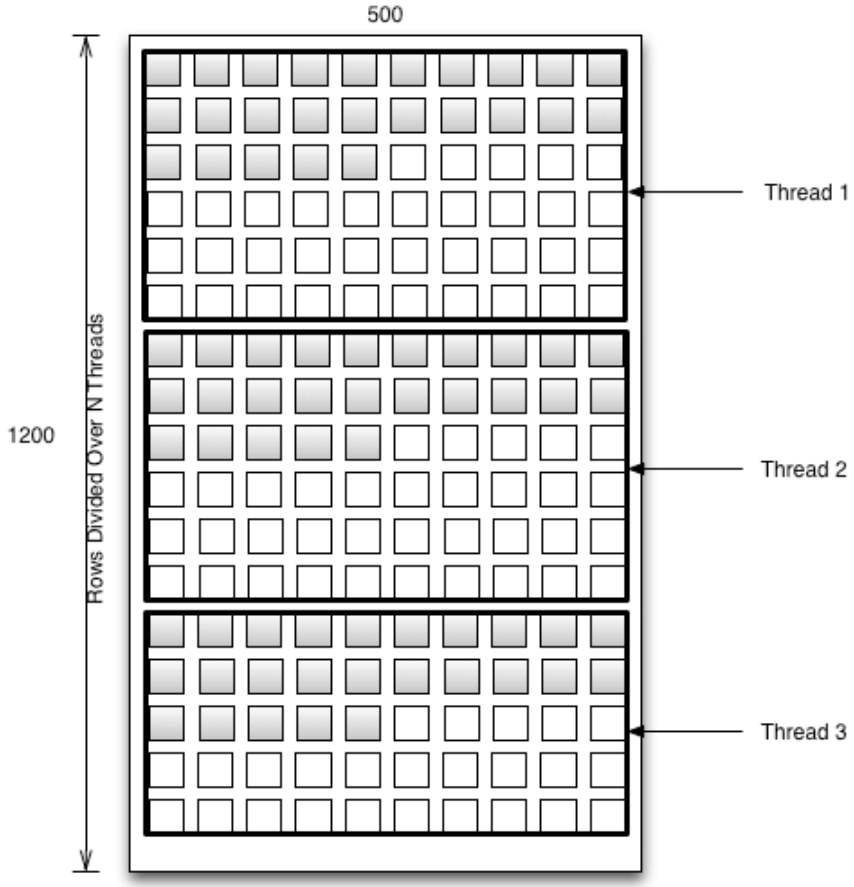
MULTITHREADED MATRIX MULTIPLICATION

Design / Implementation Decisions:

I wanted to design a generalizable algorithm for decomposing the matrix multiplication from the beginning. It seemed like this would ultimately be simpler and faster to write. The critical function in the multithreaded multiplication is multiplyChunk(n). This method takes an integer representing the number of the 'chunk' it should multiply. The starting and ending row indices of the final matrix are then calculated, and the multiplication is performed for that subset of rows in a similar manner as the single threaded multiplication.

The program takes no command line arguments. For simplicity, I decided to simply output the results of the multiplication for all 1-8 thread runs.

Diagram:



Verification:

I verify that the multithreaded multiplication algorithm gave the correct result for each run by calling a function, compareMatrices(), after each multiplication. This function simply compares the result of the multithreaded multiplication (matrix C) to the result of safe, simple, single-threaded multiplication (matrix C1). If any differences between the two matrices is found, the program prints a corresponding error message.

Additionally, a function printParts() is supplied which prints out random rows of the matrices C and C1 for visual comparison and verification.

Partners:

I worked solo on this project. All design, coding, and documentation were done by me.

Test Cases:

Sample program output, with printParts() uncommented for visual comparison:

```
Matrices Initialized!
Threads        Seconds
1               4.188014s
2               2.021662s
C:                    C1:
188700000             188700000
188900000             188900000
189100000             189100000
189300000             189300000
189500000             189500000
189700000             189700000
189900000             189900000
190100000             190100000
190300000             190300000
190500000             190500000
190700000             190700000
190900000             190900000
              No errors detected!
3               2.051076s
C:                    C1:
188700000             188700000
188900000             188900000
189100000             189100000
189300000             189300000
189500000             189500000
189700000             189700000
189900000             189900000
190100000             190100000
190300000             190300000
190500000             190500000
190700000             190700000
190900000             190900000
              No errors detected!
4               2.009314s
C:                    C1:
188700000             188700000
188900000             188900000
189100000             189100000
189300000             189300000
189500000             189500000
189700000             189700000
189900000             189900000
190100000             190100000
190300000             190300000
190500000             190500000
190700000             190700000
190900000             190900000
              No errors detected!
5               2.014170s
C:                    C1:
188700000             188700000
188900000             188900000
189100000             189100000
189300000             189300000
189500000             189500000
189700000             189700000
189900000             189900000
190100000             190100000
190300000             190300000
190500000             190500000
190700000             190700000
190900000             190900000
              No errors detected!
```

...(continued from previous page)

```
6               2.130337s
C:                      C1:
188700000               188700000
188900000               188900000
189100000               189100000
189300000               189300000
189500000               189500000
189700000               189700000
189900000               189900000
190100000               190100000
190300000               190300000
190500000               190500000
190700000               190700000
190900000               190900000
              No errors detected!
7               2.015689s
C:                      C1:
188700000               188700000
188900000               188900000
189100000               189100000
189300000               189300000
189500000               189500000
189700000               189700000
189900000               189900000
190100000               190100000
190300000               190300000
190500000               190500000
190700000               190700000
190900000               190900000
              No errors detected!
8               2.065703s
C:                      C1:
188700000               188700000
188900000               188900000
189100000               189100000
189300000               189300000
189500000               189500000
189700000               189700000
189900000               189900000
190100000               190100000
190300000               190300000
190500000               190500000
190700000               190700000
190900000               190900000
              No errors detected!
```

These results are what I expected - the multithreaded multiplication works properly, there is a noticeable performance increase from using multithreading, and no errors are detected.