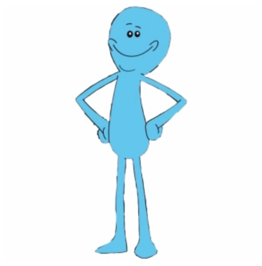
CS302

Assignment 4: Multiplication in parallel using threads



Description

For this assignment, you will multiply two numbers together in parallel using threads. The input for the program will be two numbers separated by a whitespace. You need to store these two numbers into two vectors of type `short`, so you may need to read in as a string or by character and convert each character into an integer and store into a `vector` object. When you have this portion completed, you will have two vectors and the index of the digit denotes the place value of the number, for example if I have two numbers 38961524 and 345761, then your vectors would look like the following

number1 =

3	8	9	6	1	5	2	4
[0]	**[1]**	**[2]**	**[3]**	**[4]**	**[5]**	**[6]**	**[7]**

number2 =

3	4	5	7	6	1
[0]	**[1]**	**[2]**	**[3]**	**[4]**	**[5]**

Contents of main

Using these two numbers contained in these vectors, you will need to do long multiplication (like you did in grade school). Except you will do the multiplications in parallel, so for example, one thread will multiply the digit 5 to all of the digits in number1, another thread will multiply digit 6 with all of the digits of number1, and so on. You will join the threads if either you maxed out the number of threads or if all digits of number2 have been multiplied to number1; if after joining all the threads and there is still more work to be done, then keeping spawning threads and assign each thread a digit from number2 to multiply to number1. You will need a `vector` of type `thread` to maintain all the threads that are currently running.

After each thread finishes its job, it will push its result onto a 2D vector that maintains all the partial results, this is a shared variable so you may need to use the `mutex` locking and unlocking functions to make sure any race conditions are handled. This partial results 2D vector and number1 and number2 can all be global vector objects. Once you have all the partial results in your 2D vector and all of your threads are done, you will do long addition using these partial results, you won't need to multi parallelize this, so you can do this in main without spawning threads.

Specifications

- Comment your code and your functions
- Make sure you never spawn more threads than your system can handle, use `thread::hardware_concurrency()` function to determine the amount of threads that can be runned on whatever system you're using
- Make sure your output has all the leading zeros removed

- **IF YOU ARE GOING TO USE THE SCHOOL SERVER TO RUN YOUR CODE, DO NOT USE BOBBY.CS.UNLV.EDU OR SALLY.CS.UNLV.EDU!!! PLEASE USE CARDIAC.CS.UNLV.EDU!!! THANKS!!!**

Example Output

```
$ g++ -std=c++11 -pthread ParallelMult.cpp
$ ./a.out

Enter number1: 345264
Enter number 2: 75611127480

The product is: 26105800318254720

$ ./a.out

Enter number1: 0
Enter number 2: 546376384

The product is: 0

$ ./a.out

Enter number1: 4474393485439503485904385430 9
Enter number 2: 435439589439058430958439058435 84

The product is: 19483280622885750654740026250854845912176961367572874384 03456
```

Written Portion

Additionally to your source code, you will be responsible for a 1 page write up, where you want to explain the following

- What system/compiler was used to run your program
- How did you implement the multi-threaded algorithm
- Does parallelization speed up the problem?
- Does adding more threads to the problem always cause a speed up? When does it and when does it not?
- Which parts of your code can be parallelized? Could the portion of your code that might not have been parallelized become parallelized?

Submission

Upload your source file using the following naming convention `LastName_FirstName_A4.cpp` and your write up to the moodle site by the deadline