Project 06

Goals

This project is an exploratory project, where you will finish implementing a part of an algorithm, and then explore why the algorithm performs the way it does. You will need to use either Google drive's excel program or Microsoft's excel program to perform some basic analysis on the performance of an algorithm. The last thing this will do is help research why something goes wrong, which is something you will have to learn to do in both later on in your degree as well as in the workplace.

Description

In this project you will be implementing a specific portion of merge sort, as well as performing some analysis on the parallel vs sequential performance of the algorithm. There are two portions of this algorithm

1) The sequential algorithm
2) The parallel algorithm

In this case, both algorithms will be using a **value returning** implementation, meaning we are not directly modifying the input array, and are instead returning the input array in a corrected format.

On top of this, the parallel algorithm will be running with a Threshold that will tell the algorithm weather or not to create a thread or not. This threshold is input to the algorithm form the function. The parallel algorithm two std::async calls which return a std::future<std::vector<Sortable>> type. This type can then be retrieved to get the std::vector<Sortable> underneath.

As the programmer, you will need to implement the Merge portion of this algorithm that will create a std::vector<Sortable> that will merge the Left and Right vectors into a sorted vector.

Once you have an algorithm that correctly sorts things in order, you will then move to the analysis portion of this assignment.

Running your solutions

There are two scripts.

1) verify.sh : This script will build and execute the test executable. This is used to verify your algorithm implementation is correct. This will also build your correct executable.
2) benchmark.sh : this script will build and run the actual benchmarks that you will use for your analysis. This will also build the appropriate executable.

There are no actual "Blackhawk" test programs for you to run on this program. You must simply use the executable to verify if your algorithm is correct or not. Both main.cpp files have a function that will verify your sorted array after it's called in either the sequential or parallel implementation.

For the benchmark, there will be a pretty hefty system load created. Because of this, we won't be running things on Blackhawk and will instead be running things on the JETSON cluster.

Run the verify.sh script on Blackhawk. The Benchmark script will not work on Blackhawk.

Accessing Jetson

To access the Jetson cluster, begin by logging into Blackhawk normally.

From here you will type the following

ssh jetson

This will prompt you with some authentication question the first time you log into the system. Just type "yes". It will then prompt you for your password. Type in your regular Blackhawk password.

From here you will be in a jetson subdirectory. Type "cd" to move back to your baseline linux directory. You can then navigate to your regular directory to execute the project.

Running the Benchmark on Jetson

Once you have verified your program's correctness on Blackhawk, you are now ready to run and execute the benchmark on Jetson.

To execute the benchmark, run the benchmark.sh script. This will ONLY work from jetson, and your working directory must be the same as your code.

You can track the progress of your job by typing "*squeue*". If you need to cancel a job you queued, type "*scancel ######*" replacing #s with your job ID (example: "*scancel 15819*")

This will take some amount of time; I recommend giving it at least 15 minutes to run. After you run this, you will have a series of .txt files appear, and a STDIN.o##### file appear. This 2nd file is command line output of the script execution, while the .txt files are the outputs of each phase of the executable.

What this benchmark is doing is running the main.cpp with a variety of vector sizes and providing a threshold on when to turn on threading. This threshold increases and is represented in the sort_######.txt files as the #####. An array larger than this will have its merge sort ran in parallel using the std::async, and arrays smaller will use the sequential implementation. A total of 6 different thresholds are used. **IT IS EXPECTED FOR THE SMALLER THRESHOLDS TO FAIL AT A CERTAIN POINT YOU WILL NEED TO EXPLAIN WHY.**

Deliverables

You have a two-part deliverable.

1) You are to complete the Merge function as described above. This will take the left and right vector and merge them in order. *Remember you are returning a new std::vector<Sortable> and NOT modifying either of the vectors* (Hence the const ref in the parameters)
2) You will then run benchmark script. After this is done, you need to take the data collected and do the following:
    a. Plot the time output for each threshold across the array sizes and compare the sequential to the parallel implementations.
    b. Explain why certain thresholds failed to execute after a certain point.
    c. Explain why there are speedups at the higher thresholds
    d. Explain why the smaller thresholds are so slow.
    e. This process is running on a 4 thread ARM system. What type of performance gains would you expect with this running on a modern intel i7? What about a server similar to Blackhawk running a 40+ core xeon?
    f. What are some things that may be done to improve performance further?

In part 2 you can use excel, google docs, matlab, or whatever your preference is, but I want to see graphs of some kind. Please be sure to label these plots so that its identifiable as to what they are. As far as parts B-F, I expect the answers to be "as long as necessary". There's no particular length but be sure to answer the questions to your best ability.

Final notes

Please do NOT try to run the main executable on Blackhawk. This is designed to use multiple threads and if multiple people try to use this at the same time on Blackhawk there will be very inconsistent results as well as cause other things to slowly lock up on Blackhawk for other people, potentially causing your account to also get locked out. Also be sure that you have ran the verify.sh script BEFORE trying to run benchmark.sh and you have verified your solution is correct there.