CS302

Assignment 4: ThreadSort

Description

For this assignment, you will use a set of threads to sort a given array. For the input you will read in two integers: an integer to denote the list of the list which we will denote as `listSize` and the amount of threads we wish to use which we will denote as `numThreads`. You will re-prompt for input if the numbers read in are either non positive, not a numerical type, or if the amount of threads read in exceeds the amount of threads your system can handle. You can assume both the `listSize` and `numThreads` values will both be powers of 2, just to make life a lot easier.

ThreadSort Algorithm

1. Allocate a vector of `listSize` number of items and assign a random number into each element between 1 and `listSize`, declared this vector as a global vector, to make things simple
2. Create exactly `numThreads` amount of threads and each thread will run insertion sort on each `listSize / numThreads` sublist, you will need to pass in the correct left and right indices (endpoints) into each thread function such that each thread is assigned its own list without overlapping the other threads, the insertion sort prototype can be seen below

   ```
   void insertionSort(int left, int right);
   ```

3. After step 2, we now have exactly `numThreads` amount of sorted sublists each of size `listSize / numThreads`, we must now perform the steps to merge these sublists to form larger sublists using the following function that will be assigned to a thread

   ```
   void mergeLists(int leftLeft, int leftRight, int rightLeft, int rightRight);
   ```

4. You will now need to merge each adjacent pair of `listSize / numThreads` size sorted sublists, thus you will need to spawn `numThreads / 2` amount of threads (you will need to compute the 4 endpoints for each thread to tell it which two sublists it needs to merge)
5. After step 4, you perform the same task except you will spawn half as many threads each time which will sort two larger pair of adjacent sublists (each sublist will double in size each time), and you continue this process until you have one thread that merges two adjacent pair of sublists which are both `listSize / 2` in size
6. Once the list is sorted and all threads are done, output the list unless the `listSize` is too large, anything larger than 512 elements would be too large

So for example, if we have a vector of 65536 and have 8 threads, we will spawn 8 threads and each thread will sort 8912 elements (since 65536 / 8 = 8912), and then we will spawn 4 threads and each thread will merge two adjacent lists of size 8912 into arrays of 16384 (there will be 4 of these). Then we will spawn 2 threads to merge the 4 sorted lists into 2 sorted lists, and then we will spawn 1 thread which merges two lists of size 32768 and then you will have one sorted array of size 65536.

The difficult part will be computing the new ranges for each thread each time such that no two threads overlap, i.e. will not have multiple threads accessing the same element in the vector and for spawning the correct amount of threads each time. Your code must be scale-able to the amount of threads you have and the size of the list. You may use a global vector to store the numbers list, do not use reference parameters with threads since thread functions do not like that.

Write Up

- Tabulate the runtimes (in real time) for various amounts for list sizes (all powers of 2) along with various amount of threads
- Using the same list sizes from the above portion, tabulate the runtimes (in real time) for a sequential sort (you can use quicksort or mergesort)
- Conclude which method was faster (sequential or parallel) and explain why you obtained those results
- Hypothetically if we have `listSize / 2` amount of threads that could run in parallel, would our parallel sorting algorithm have a better runtime than a sequential sorting algorithm?

Specifications

- Comment your code
- Make sure you never spawn more threads than your system can handle
- You may use a global vector to store your vector, but the parameters passed into each thread function must be defined in main and not be globals since the amount of threads used is not predetermined
- You can assume list size and number of threads will be a power of 2

Sample Run

```
You won't have a hard time determining if it worked or not...
```

Submission

Upload your source code and write up to webcampus by the deadline

References

- Link to the top image can be found at $https://www.deviantart.com/wor-sazbat/art/I-m-Mr-Meeseeks-look-at-me-690741223$