Dee Dee McCroskey

4/21/18

Quicksort in Open MP

I decided to write and run a parallel version of quicksort for my final project. My program is made up of basically two parts. The parallel versions of quicksort and the non-parallel version.

Here is the non-parallel version of quicksort.

void quickSort(int\* a, int left\_index, int right\_index)

{

int left;

int right;

int pivot;

if(left\_index >= right\_index)

return;

left = left\_index;

right = right\_index;

// pivot selection

pivot = a[(left\_index + right\_index) /2];

// partition

while(left <= right) {

while(a[left] < pivot)

left++;

while(a[right] > pivot)

right--;

if(left <= right) {

swap(a,left,right);

left++;

right--;

}

}

// recursion

quickSort(a,left\_index,right);

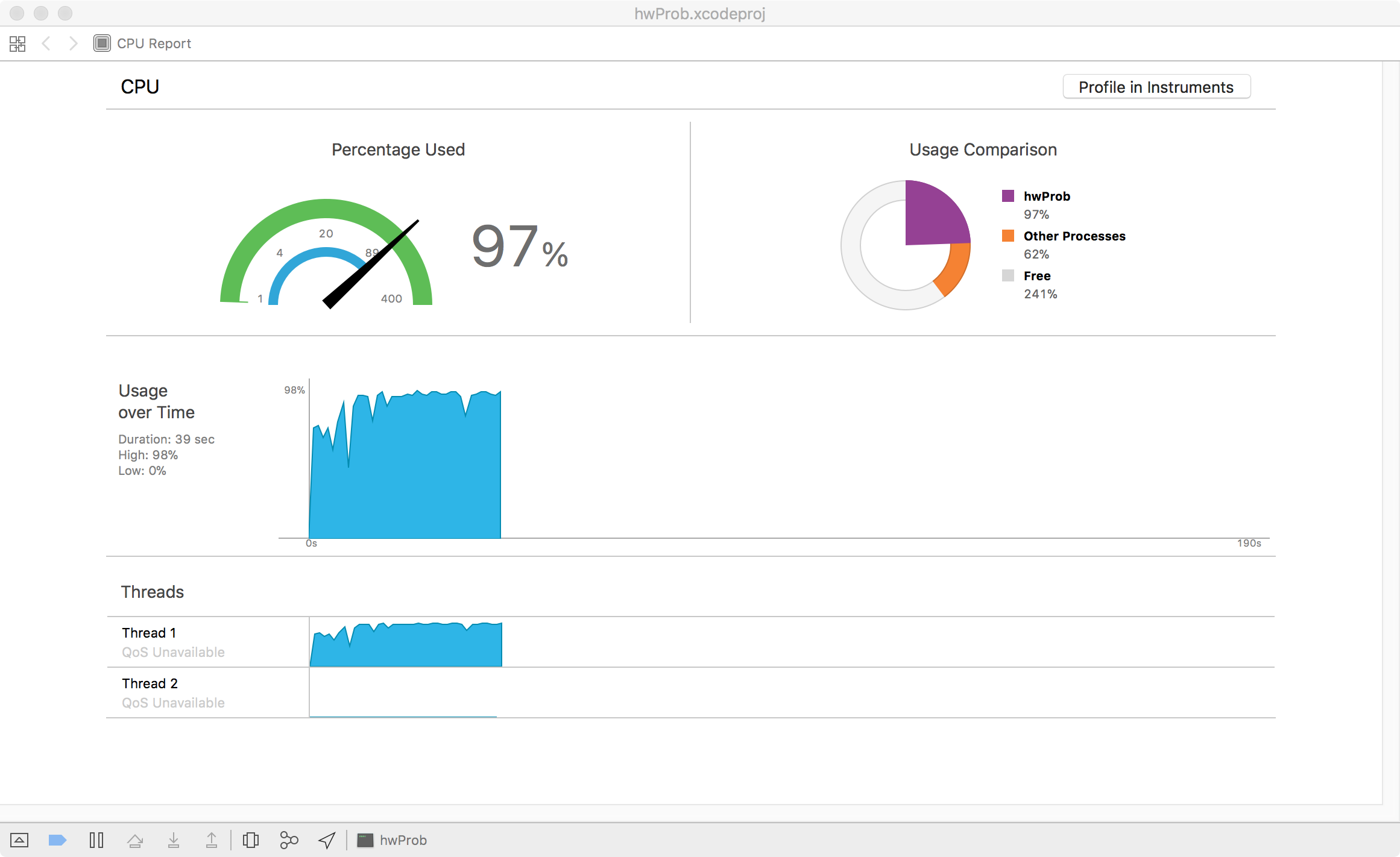
quickSort(a,left,right\_index);

}

This version of the sort is a recursive version that uses a middle pivot point. I found it at: <http://www.bogotobogo.com/Algorithms/quicksort.php>

When run under my test conditions it generally completed in 48 seconds.

Here is an images of it running on my machine.



The parallel version is more interesting to look at.

This is the outer section of the sort. It sets up the threads the sort will be run on. In our case here that is 8 threads.

It is also having it so this initial call to quickSortPartition() will be in its own thread. This is useful because it gives us a main thread that the threads handling the partitioning will branch from.

When run under my test conditions it generally completed in 25 seconds. Here is an image of it running on my machine

Here

void quickSortParallel(int\* array, int sizeOfArray, int numThreads)

{

#pragma omp parallel num\_threads(numThreads)

{

//identifies a section of code that must be run by a

//single available thread and avoid the implied barrier at

//the end of the single directive

#pragma omp single nowait

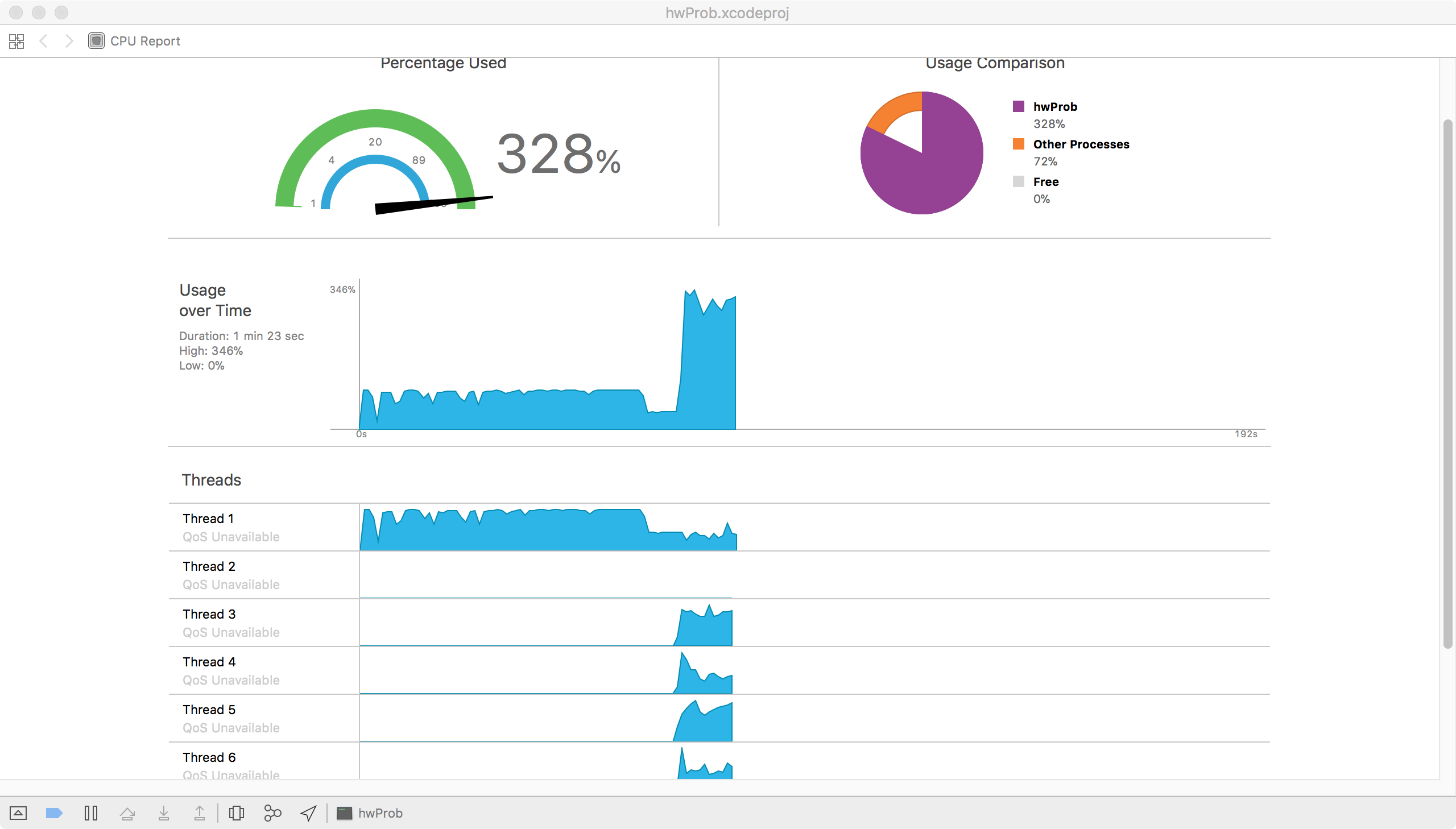
{

quickSortPartition(array, 0, sizeOfArray-1);

}

}

}



This is the important part to look at. It is generally the same as the original quicksort. It differs at the bottom when the recursion begins though. As long as there the partition is fairly large (>700) it is handled in parallel. This is done so the different segments being sorted can come together at the end. It also resolves the segment fault problems that occurred without it. If the partition is fairly large (>700) than each partition is started in its own task block. This omp pragma is starting them while letting them still have access to what is happening outside their individual threads.

void quickSortPartition(int\* array, int low, int high)

{

//set up loop variables

int lowTemp = low;

int highTemp = high;

//center point

int pivot = array[(low + high) / 2];

{

//this where the partition happens

while (lowTemp <= highTemp) {

while (array[lowTemp] < pivot) //while the left value index hasn't passed the middle pivot point

lowTemp++; //move left side toward pivot (rightward)

while (array[highTemp] > pivot) //while right side value index hasn't passed the middle pivot point

highTemp--; //move right side toward pivot (leftward)

if (lowTemp <= highTemp) {

swap(array, lowTemp, highTemp);

lowTemp++; //move left index forward

highTemp--; //move right index forward

}

}

}

if ( ((high-low) < 700) ) //if the right is not as far from the left as needed (the partition isn't that big)

{ //just continue serially otherwise you get errors

if (low < highTemp)

{

quickSortPartition(array, low, highTemp);

}

if (lowTemp < high)

{

quickSortPartition(array, lowTemp, high);

}

}

else

{

// omp task - identify a block of code to be executed in parallel with the code outside the task

// partition both sides

#pragma omp task

{

quickSortPartition(array, low, highTemp);

}

#pragma omp task

{

quickSortPartition(array, lowTemp, high);

}

}

}

Resources Used

* <http://www.bogotobogo.com/Algorithms/quicksort.php>
  + This where I got my original quicksort algorithm.
* <https://www.youtube.com/watch?v=dD4ls9cLnMk>
  + This video explains the general idea of a parallel quicksort. It was used as a warm up for my thought process.
* <https://www.geeksforgeeks.org/quick-sort/>
  + This is the quicksort I first ever learned. It was used to refresh my memory on quicksort in general.
* <http://www.openmp.org/wp-content/uploads/sc16-openmp-booth-tasking-ruud.pdf>
  + This explained the idea of using omp tasks to solve the problem of paralyzing the sorting of the low and high partitions of the array.