Merge Sort on GPU

using Compute Unified Design Architecture (CUDA)

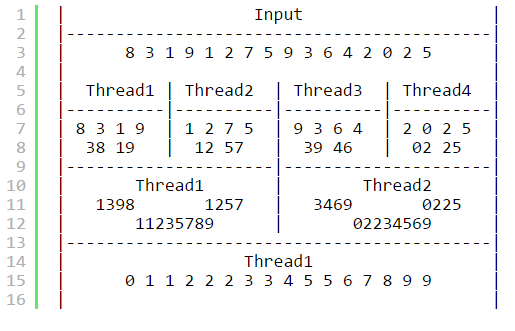
## Algorithm

I chose to try and parallelize a simple bottom-up mergesort. It can happen in just a few steps:

1. Start with two lists: Your input array, and a temp array that’s the same size.
2. Define a width, starting at 2. During each step, width gets multiplied by 2.
3. While width < 2N, sort each width-sized chunk of the list into the temp list. Then switch the pointers of the two lists. This means you can avoid allocating lots of tiny arrays, or copying temp back to input.

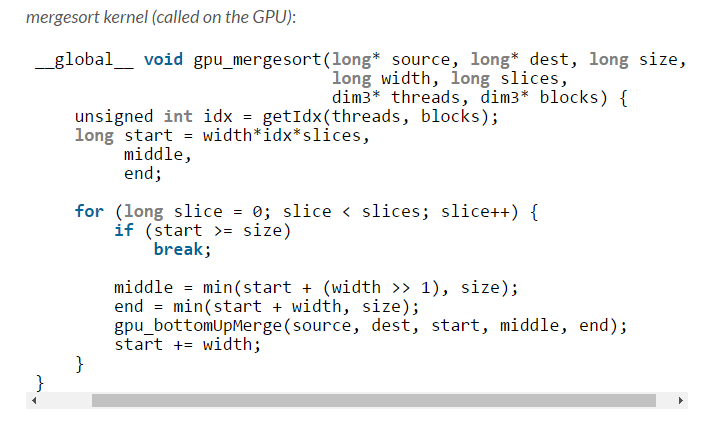
\* This is the step that happens in parallel. Each thread gets a chunk of the list to sort.  
\* Two halves of each chunk are are sorted / merged against each other into the temp array.  
4. You end up with one big chunk being sorted into the final list, and you switch input and temp one last time, returning temp

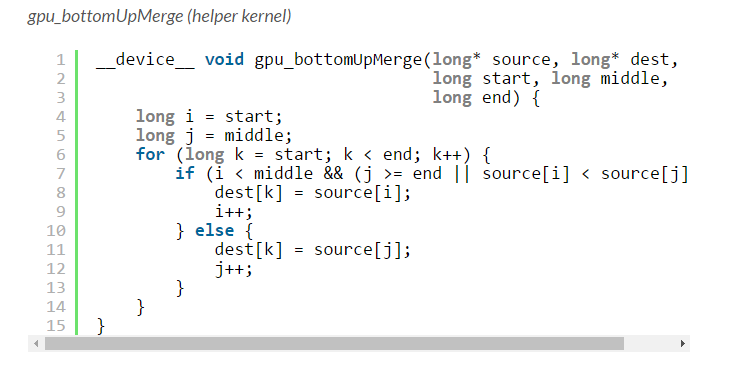
Here’s what it looks like with 4 threads sorting 16 numbers (notice that while (N / width) > number of threads, the time complexity becomes O(Nlog(N)/k) where kis the number of threads):

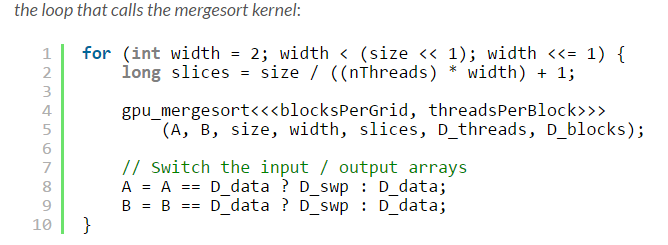


## **Implementation**

The three main functions used in the sort are:







## **Actual Performance**

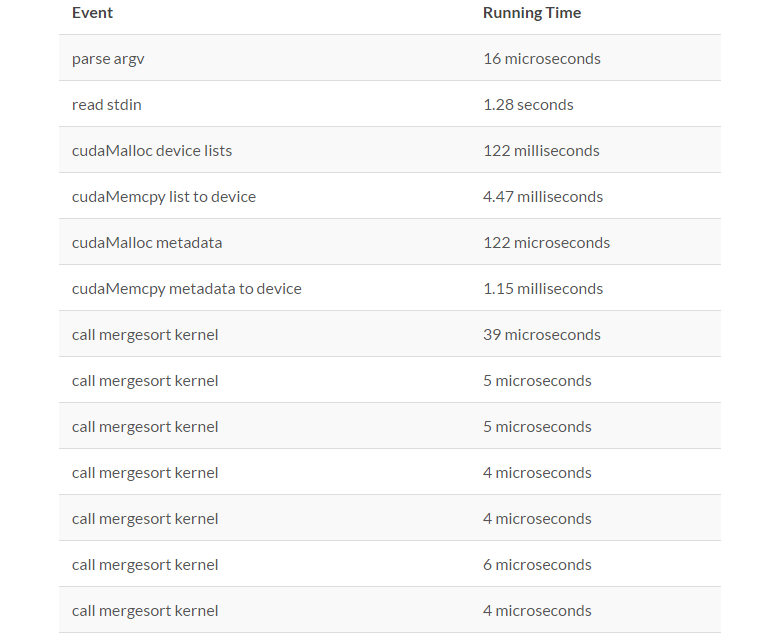
Sorting 1 million integers.

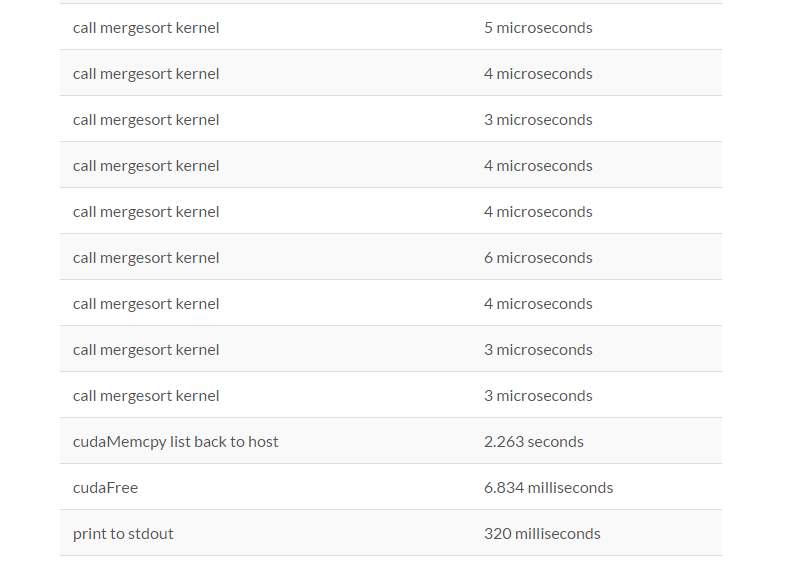
**Built-in sort**

sort -n out.txt -> 2.738 seconds

**My CUDA mergesort, with 256 threads**

./mergesort -v out.txt -> 3.871 seconds





## **Takeaway**

* Calling a GPU kernel is very cheap. Allocating / streaming data is not
* Don’t use too many threads or your screen will freeze indefinitely
* Simple GPU programming with CUDA isn’t too difficult