This is my solution for the parallelisation of the radix sort algorithm.
To run the program you can input <n> <seed> <useBits> [totalThreads] where:

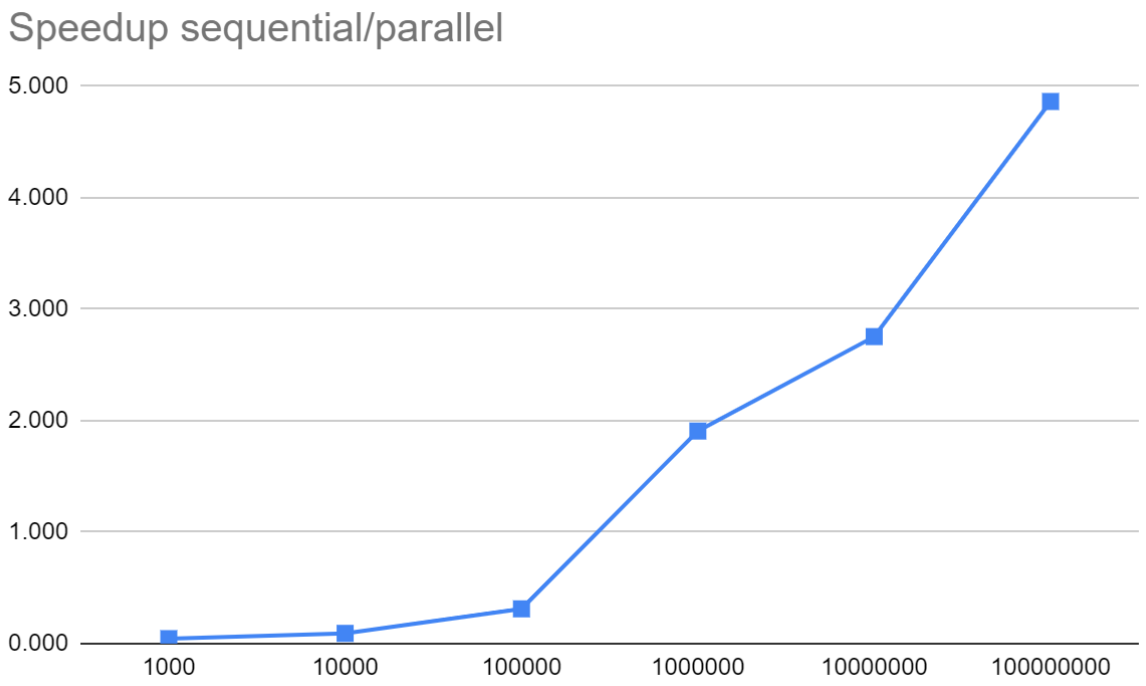
- *n* is the length of the array of numbers
- *seed* is the number used as seed for the generation of the numbers
- *useBits* is the number of bits of each bucket
- *totalThreads* is the number of threads used, this parameter is optional, if it's missing the number of threads of the pc is set

I begin finding the max number in parallel; each thread finds its local max and then merges the result finding the greatest of the max numbers.
Then I parallelise the counting sort part; each thread has a part of the total array and compute this part of the array as if it was all the array.
The counting sort is divided in 3 steps, so, at the end of each step I do some operation to save the local data in the shared variables.

Here you can see the result of my solution:

n	java	sequential	parallel
1000	0.379	0.153	3.431
10000	0.862	0.351	3.826
100000	9.741	1.387	4.447
1000000	56.589	17.627	9.244
10000000	621.027	137.248	49.832
100000000	7181.944	2099.973	431.779

Times in milliseconds



As we can see the parallelization can be useful when we have more than about 500000 numbers in the array.
After this number we have a really good speed up.

So, as it is common, it is useful to parallelise this algorithm but only if you have to sort big arrays.