Sergi Aliaga

Northeastern University

Homework 1

Question 2

In this problem, you will start by selecting a sorting algorithm to sort 7,500 random integers with values 1-1,000,000. We will use pthreads in the problem to generate a parallel version of your sorting algorithm. If you select code from the web (and you are encouraged to do so), make sure to cite the source of where you obtained the code. You are also welcome to write your own code from scratch. You will also need to generate the 7,500 random integers (do not time this part of your program). Make sure to print out the final results of the sorted integers to demonstrate that your code works (do not include a printout of the 7,500 numbers). Provide your source code in your submission on Canvas.

a. *Run your program with 1, 2, 4 and 8 threads on any system of your choosing. The system should have at least 4 cores for you to run your program on. Report the performance of your program.*

For this section of the assignment, I implemented a parallel version of the Merge Sort algorithm (code borrowed from https://malithjayaweera.com/2019/02/parallel-merge-sort/ and included in Q2.c). Figure 1 reports the results obtained for ten runs of the algorithm with the suggested numbers of threads.

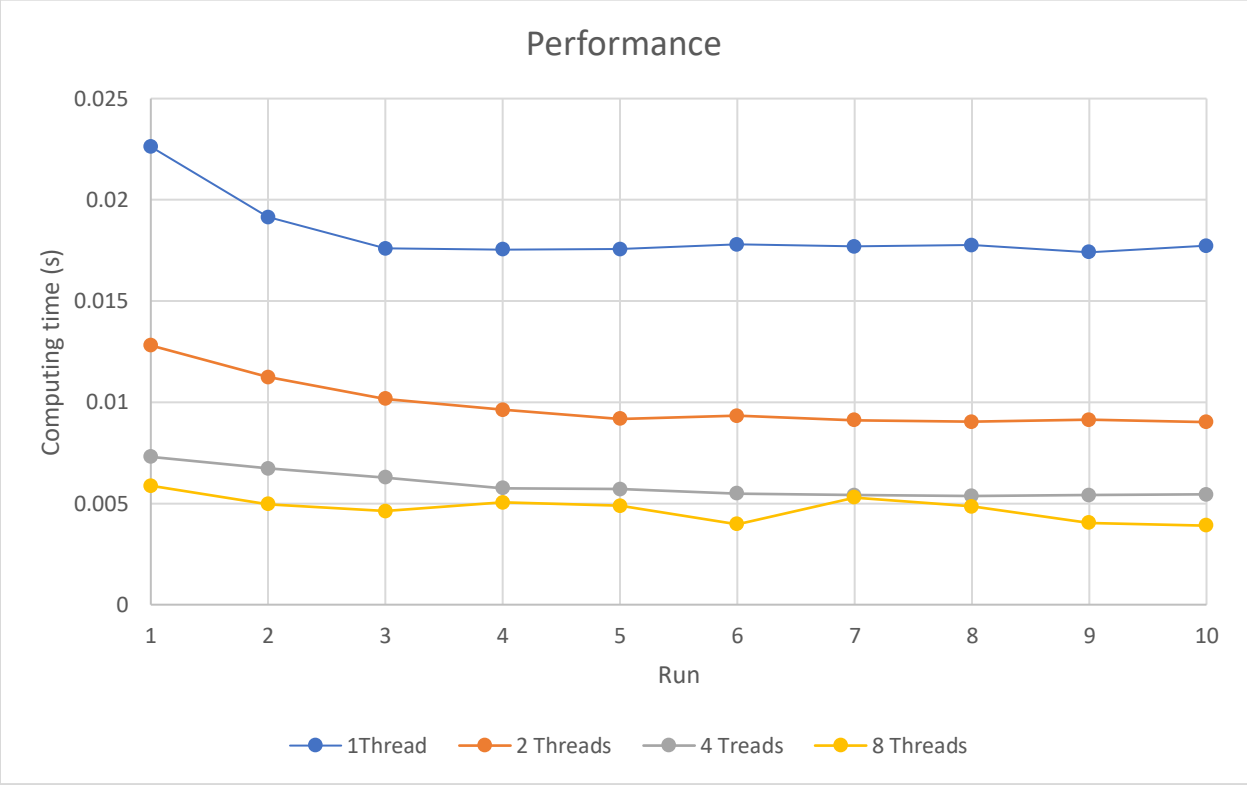

Figure 1: Performance of the parallel merge sort implementation

Table 1 displays the obtained average values. We observe that, in this particular setup (sorting algorithm and size of the input data) the benefits of parallel computing are clear. However, we observe that the performance between the implementation with 4 and 8 threads are not that

different, compared to the previous increments. This is because the more threads we spawn, the closer we are to the tradeoff between number of threads and overhead added for the multithreading (Amdahl's Law). If we increased the size of the input as well as the number of threads, we might not observe this behavior (Gustafson's law).

Threads	1	2	4	8
Avg. time(s)	0,018	0,0099	0,0059	0,0047
Speedup (%)	-	**46**	**68**	**74**

b. *Describe some of the challenges faced when performing sorting with multiple threads.*

At first, I tried to implement a parallel version of the merge sort algorithm without relying on external code. The major problem I could not get over with is the fact that "*pthreads_create"* only accepts one argument for the function assigned to the created thread. This forced me to play around with structs in order to provide the function in "*pthreads_create"* with more than one argument. That lead to a long learning session on pointers in c, which refreshed a lot of concepts of c programming but couldn't manage to make my code work in time. Eventually, I looked for an implemented version on the Internet and learnt its way to implement the parallel merge sort.

One particularly challenging idea to understand has been the fact that for parallel merge sorting, there is no need to use locks, since each thread access an independent portion of the array and the final array merge is done sequentially. The function in charge of that last merging, *"merge_sections_of_array",* has also been particularly challenging to understand, due in part by its recursive nature.

c. *Evaluate both the weak scaling and strong scaling properties of your sorting implementation*

As outlined in the first section of this question, when reaching 8 threads we observe that the increase in performance w.r.t. 4 threads is not that noticeable. I can therefore say that the strong scaling of my algorithm implementation is reaching its limit at around 8 threads. On the other hand, I looked at the weak scaling properties of the implementation by increasing the size of the array accordingly with the number of threads. The results are reported in Figure 2, where the size of the input has been doubled each time the number of threads doubled too. This time we observe a similar performance until 4 threads are reached, which means that the algorithm weakly-scales well. However, when reaching 8 threads, we observe a reduction in performance, which might indicate that 8 threads is also the limit for weak scaling of this implementation.

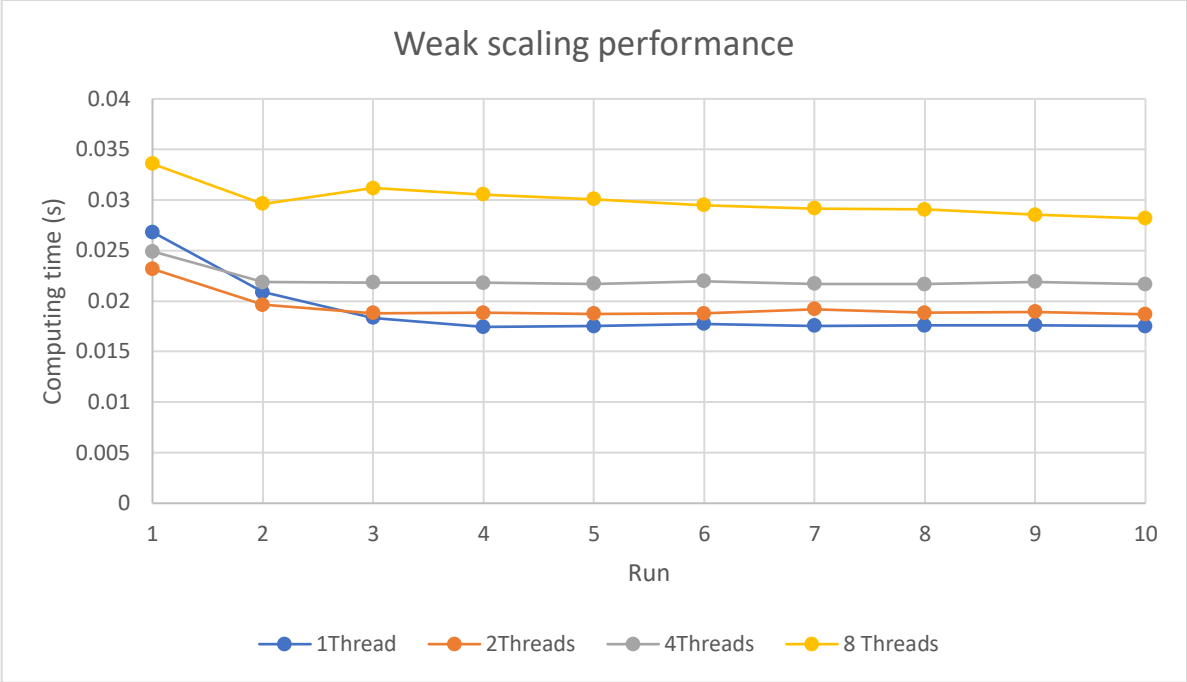

Figure 2: Weak scaling performance of the parallel merge sort implementation