Parallel and Distributed Programing Assignment 3

Fredrik Gustafsson

June 2, 2020

1. Implementation

The parallel Quicksort algorithm was implemented by starting to divide the initial data into n chunks, after which each PE localy sorted their chunk using a prebuilt sequential quicksort. For the global sort the pivot was chosen within each group by the PE with rank 0. Once the number of elemetns to be sent had been calculated the PEs within each group exchange them after which we split the communicator into two parts. To keep track of how many time we repeat the global sort a variable loc_size where used which was divided in each iteration until reaching the number 1.

2. Performance Meassurements

To meassure the performance of the program a start time meassurement was taken after the reading of the file and an end meassurement was taken after the gatering into a final sorted array. The code was then run on the rackham cluster.

2.1. Strong Scaling

To meassure the strong scaling, the speedup from increasing the number of PEs, one, two, four, eight and 16 PEs were tested on three input files containing $1.25 * 10^8$, $2.5 * 10^8$, and $5 * 10^8$ integers for each pivoting strategy. The results are shown in table 1-8 in the appendix. Plots of achieved speedup together with the theoretical speedup for each problem size is shown in Figure 1-3 below.

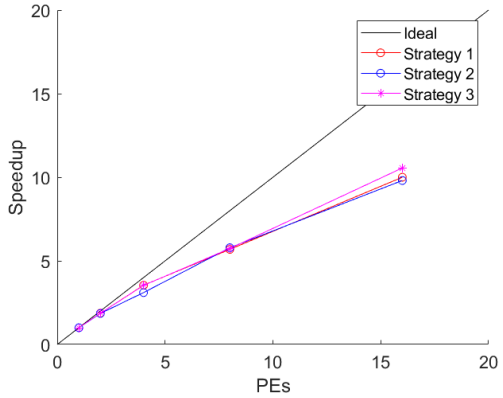

Figure 1: Speedup when using one, two, four, eight and 16 PE on array with $1.25 * 10^8$ numbers.

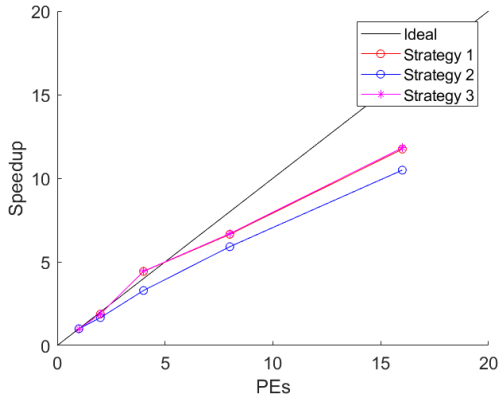

Figure 2: Speedup when using one, two, four, eight and 16 PE on array with $2.5 * 10^8$ numbers.

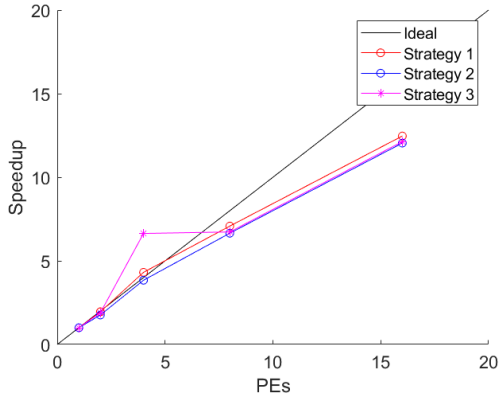

Figure 3: Speedup when using one, two, four, eight and 16 PE on array with $5*10^8$ numbers.

Finally we also investigate the strong scaling using a sorted array, $1.25{*}10^8$ integers, but where the numbers are in descending order. The results are shown in table 9-11 in the appendix and in Figure 4 below.

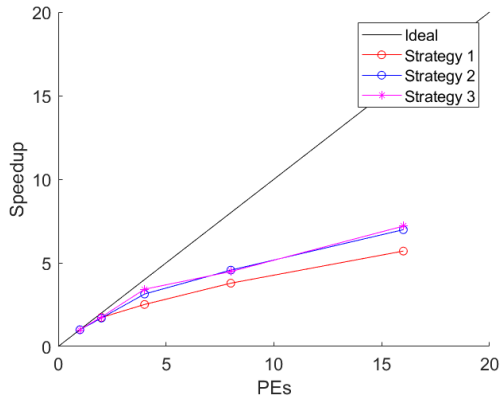

Figure 4: Speedup when using one, two, four, eight and 16 PE on array with $1.25 * 10^8$ numbers sorted in descending order.

2.2. Weak Scaling

To investigate the weak scaling we once again investigate how different number of PEs affect the computation but this time when varying the problem size. The omplexity of Quicksort is on average $O(nlog_2(n))$ hence if the problem size is doubled then we should also double the number of PEs, under the assumption of theoretical weak scaling. The time for each pivot strategy is shown in table 12-14 in the appendix and summarized in Figure 5 below.

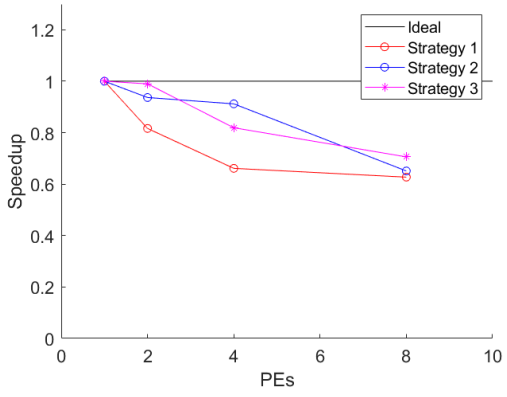

Figure 5: Weak scaling using one, two, four, eight, and 16 PEs.

3. Discussion

From Figure 1-3 we can see that the program clearly does not follow the theoretical speedup after eight PEs. We can also conclude that the second pivot strategy consistently has the worst strong scaling. On the other hand pivot strategy one and three manages to achieve super linear speedup at four PEs for the two larger arrays, $2.5 * 10^8$ and $5 * 10^8$. This might be due to them having a better balanced load thereby being able to store the data in cache memory. Furthermore it appears as the strong scaling becomes better with the larger datasets. When instead looking at the descending ordered data, Figure 4 we can see that non of the pivot strategies achieves a good strong scaling but in this case strategy one has the worst performance. On the other hand comparing the time used by the algorithm, 1-2 with 9-11 we can see that the actual time spent used by the program when the data is ordered is smaller than when the data has a random order which might be due to the o3 optimisation flag.

Lookig at the weak scaling show us that pivot strategy two and one has the worse weak scaling compared to strategy 3, with the exception of four PEs. A likely reason for this is workload imbalance since after the first iteration we do not guarantee that each PE has the same amount of data which in turn could affect the time used when each PE calculates its median, when calculating the pivot element, or lead to some PEs being able to store the data in cache while other are unable to do so. So judging by that it appears as pivot strategy three is best at achieving load balance compared to the other strategies. Another reason for the lower speedup is communication overhead, due to the need to pass values between PEs during each iteration. However when using the profiler tool the, parallel, part of the program which takes the most time is the splitting of the communicator with MPI_Comm_split which is then probably what leads to the significant decrease in speedup.

4. Appendix

4.1. Strong scaling meassurements

Number of PEs	Time
1	22.80
2	12.20
4	6.44
8	4.01
16	2.28

Table 1: Time for different number of PEs on dataset consisting of $1.25 * 10^8$ integers using pivot strategy 1.

Number of PEs	Time
1	23.01
2	12.32
4	6.52
8	4.01
16	2.18

Table 2: Time for different number of PEs on dataset consisting of $1.25 * 10^8$ integers using pivot strategy 3.

Number of PEs	Time
1	54.83
2	29.17
4	12.39
8	8.25
16	4.67

Table 3: Time for different number of PEs on dataset consisting of $2.5*10^8$ integers using pivot strategy 1.

Number of PEs	Time
1	48.45
2	29.15
4	14.75
8	8.20
16	4.62

Table 4: Time for different number of PEs on dataset consisting of $2.5*10^8$ integers using pivot strategy 2.

Number of PEs	Time
1	54.96
2	29.19
4	12.38
8	8.22
16	4.65

Table 5: Time for different number of PEs on dataset consisting of $2.5*10^8$ integers using pivot strategy 3.

Number of PEs	Time
1	118.93
2	60.85
4	27.57
8	16.80
16	9.54

Table 6: Time for different number of PEs on dataset consisting of $5*10^8$ integers using pivot strategy 1.

Number of PEs	Time
1	114.17
2	64.81
4	29.72
8	17.16
16	9.48

Table 7: Time for different number of PEs on dataset consisting of $5*10^8$ integers using pivot strategy 2.

Number of PEs	Time
1	115.26
2	60.75
4	27.36
8	17.13
16	9.50

Table 8: Time for different number of PEs on dataset consisting of $5*10^8$ integers using pivot strategy 3.

Number of PEs	Time
1	7.53
2	4.35
4	3.00
8	1.99
16	1.32

Table 9: Time for different number of PEs on dataset consisting of $1.25 * 10^8$ integers in descending order using pivot strategy 1.

Number of PEs	Time
1	8.31
2	4.90
4	2.65
8	1.82
16	1.19

Table 10: Time for different number of PEs on dataset consisting of $1.25 * 10^8$ integers in descending order using pivot strategy 2.

Number of PEs	Time
1	8.42
2	4.78
4	2.46
8	1.88
16	1.17

Table 11: Time for different number of PEs on dataset consisting of $1.25 * 10^8$ integers in descending order using pivot strategy 3.

4.2. Weak Scaling

PEs	n	Time
1	$1.25*10^8$	21.61
2	$2.5*10^8$	26.47
4	$5*10^8$	32.68
8	$10*10^8$	34.46

Table 12: Weak scaling for pivot strategy 1.

PEs	n	Time
1	$1.25*10^8$	23.96
2	$2.5*10^8$	25.58
4	$5*10^8$	26.27
8	$10*10^8$	36.80

Table 13: Weak scaling for pivot strategy 2.

PEs	n	Time
1	$1.25*10^8$	23.20
2	$2.5*10^8$	23.44
4	$5*10^8$	28.32
8	$10*10^8$	32.87

Table 14: Weak scaling for pivot strategy 3.