Parallel and Distributed Programing Assignment 1

Fredrik Gustafsson

April 16, 2020

1. Description of Parallelisation

To parallelize the provided serial code we start with PE 0, henceforth referenced to as master, reading the data. After this we broadcast the number of values in order to allow each other PE to allocate the memory needed for the task, both receiving and sending. We can then broadcast the data pices which each Pe will work on locally. The stencil application then follows the same outline as in the serial case with the difference being that we at the start of each application have to let the PEs communicate with each neighbors to receive their data, using MPI_Isend and MPI_Irecv waiting until the send and requests are finished. Once the final number of stencil applications have been performed the master PE collects the distributed results and each PEs execution time after it finds the highest time, for reporting and then prints the resulting data to the output file. Finaly each PE frees the allocated memoryh for its data while the master also frees the allocated memory used for the total data and time measurements.

2. Performance Meassurements

To meassure the performance of the parallelisation each PE starts a timer vefore entering the application loop. Once the applications are finished, i.e after N steps, each PE takes a final time and calculate the time difference and finaly the master PE collects each time meassurement to calculate the highest time. The code was run on the Linux host Gullviva which use a AMD Opteron (Bulldozer) 6274, 2.2 GHz, 16-cores, dual socket CPU, at runtime the flag $--bind-to$ was set to none as instructed in the instructions. To try and lower fluctuations in the timing the program was executed ten times for each number of PEs used, or for weak scaling, each new number of stencil applications, i.e. the results presented is the average of ten executions.

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 using the file with 1000000 numbers and applying the stencil 10000 times. The resulting average time for each number of PEs is shown in Table 1

Number of PEs	Average Time
1	47.7594
2	17.1732
4	8.4649
8	4.1409
16	2.9531

Table 1: Average time for different number of PEs

A plot of the achieved speedup together with the teorethical speedup is shown in Figure 1 below.

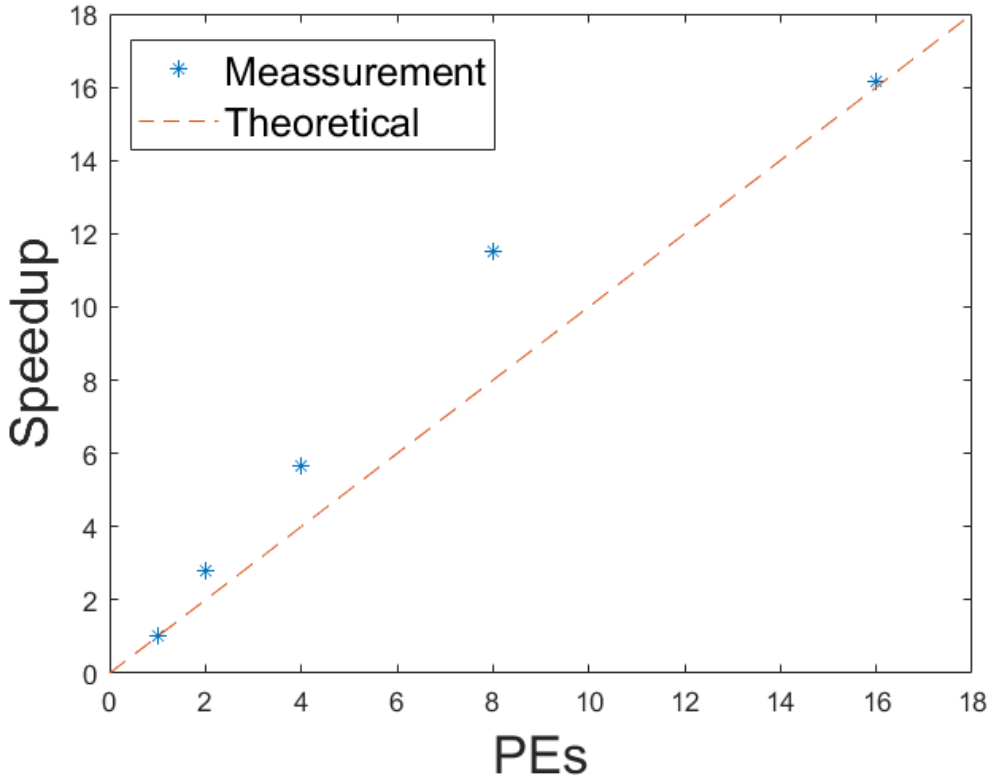

Figure 1: Speedup when using one, two, four, eight and 16 PE.

If we on the otherhand look at the speedup when setting the base to two PEs, i.e.

we compare the speedup starting from the time used by two PEs we instead get Figure 2 below.

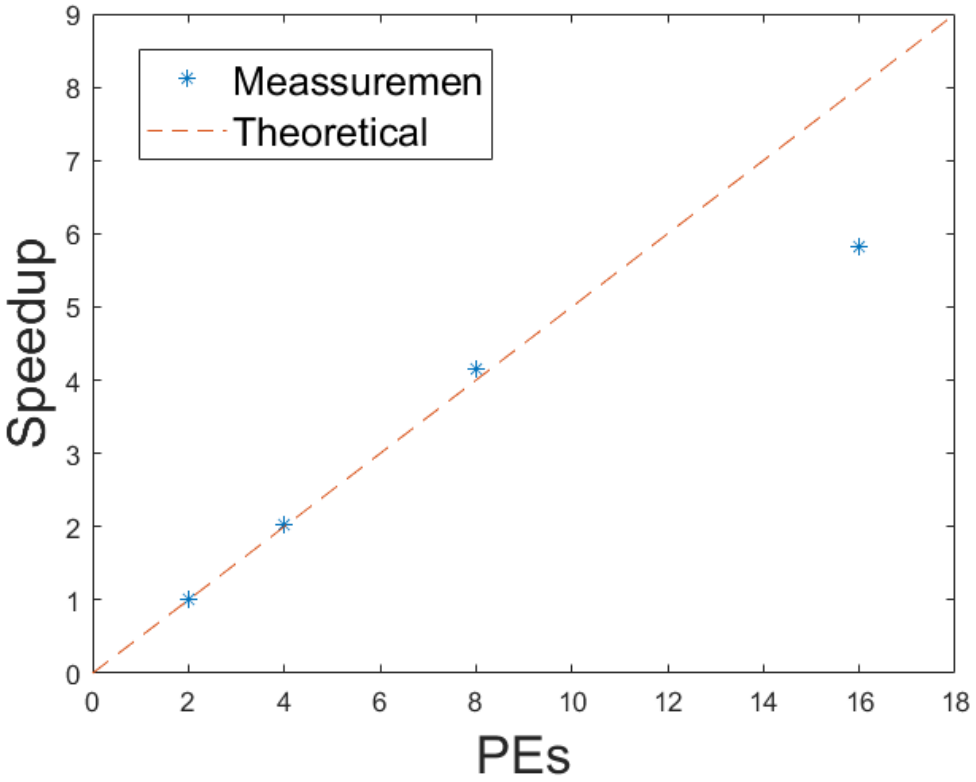

Figure 2: Speedup when using two, four, eight and 16 PE.

2.2. Weak Scaling

To investigate the weak scaling we once again investigate how different number of PEs affect the computation. However this time we also vary the number of stencil applications, increasing it at the same time as increasing number of PEs. Once again the file containing 1000000 numbers were used. The average time from 10 executions are shown in Table 2 below.

PEs	Nr. of Stencil Applications	Average Time
1	10000	47.7594
2	30000	50.6946
4	60000	50.0994
8	120000	49.0214
16	240000	50.1678

Table 2: Weak scaling of time for different amount of PEs and number of stencil applications.

3. Discussion

IF one were to look at figure 1 then the result would be quite amazing considering that two PEs resulted in a speedup of almost three and eight PEs gave a sppedup close to twelve. However when taking Figure 2 in mind it looks more realistic, four PE instead of two gave a speedup of two while eight PEs gave a speedup of four. This follows the theoretical speedup fairly well however using 16 PEs did not further halve the time, compared to using four, on the otherhand Figure 1 places the speedup of 16 PEs on the theoretical line of speedup when using one PE as the base case. One reason for this might be that the variance when using a single PE is higher and hence the ten average did not result in a fair average or that the way the communications where done resulted in a slower execution time. Table 2 show the same behaviour where we can double the number of stencil applications as we double the PEs whithout a large difference in time. However we can once again see the disparity when comparing the results of one PE with the others since we have to tripple the number of stencil applications between one and two PEs to have a similar execution time. In conclusion it appears as if the parallel implementation works wellachieveing a good speedup for the number of PEs tested.