NSSC 2

Exercise 1: MPI-Parallelization of a Jacobi Solver

Group No. 12

Aman Bhardwaj (12333472)

Florian Frech (12308544)

Task 2: One-Dimensional Decomposition

Parallel Speed Up and Efficiency

The parallel speed up is measured as the ratio of the time taken to execute the task sequentially ($T_{sequentially}$) to the time taken to execute in in parallel ($T_{parallel}$):

$$S = \frac{T_{sequential}}{T_{parallel}}$$

Ideally, the spedup is proportional to the number of processes used., i.e. if we double the number of processes, we would solve the problem in half the original time.

However, Amdahl's Law makes it clear that there is an upper bound that can be achieved by parallelizing a computation. It states that the overall fraction is limited by the fraction of the program that cannot be parallelized.

Amdahl's Law:

$$S_{max} = \frac{1}{(1-P) + \frac{P}{n_p}}$$

- S_{max}: Maximum speedup
- P: Fraction that cannot be parallelized
- N: Number of processes / threads

Efficiency in parallel computing is measured as the ratio of the speedup achieved by parallel execution to the number of processes / threads. This describes how well the parallel algorithm scales with increasing resources.

$$E = \frac{S}{n_p}$$

Similar to Amdahl's Law, an efficiency of $E = 1$ is the ideal scenario. This means that the used resources are optimally utilized, and the speedup is directly proportional to the number of processing units.

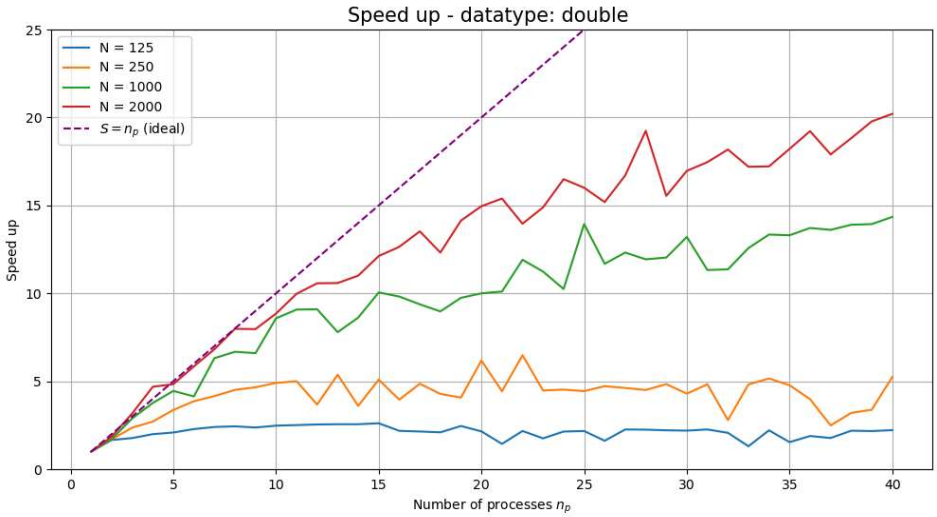

Fig. 1: Speed up of total runtime

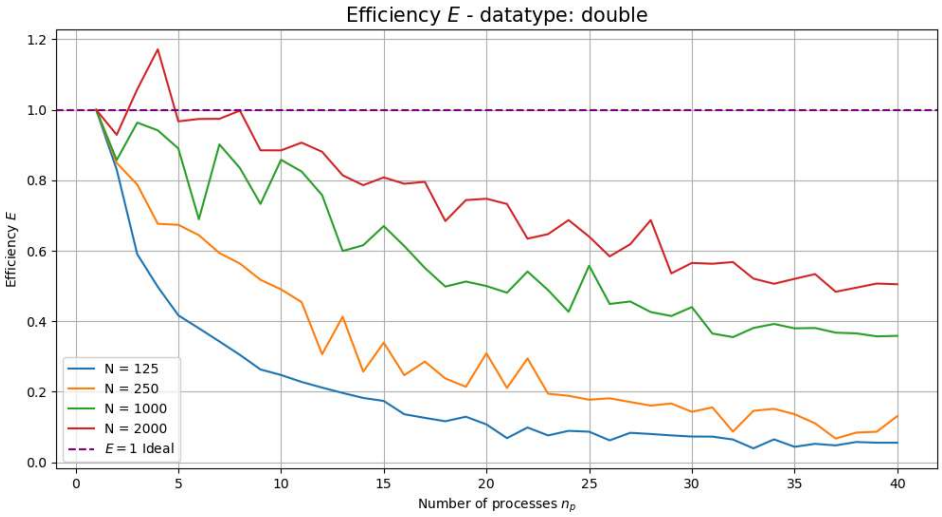

Fig. 2: Efficiency

Figure 1 and figure 2 show the speed and the efficiency for a Jacobi solver with a 1D grid decomposition for different grid sizes N^2.

A range of 1 to 40 processes were used on the cluster to perform a benchmark test.

The plots in both figures show spikes in their curves for all problem sizes. This is likely due to the decomposition of the grid.

The grid is decomposed in a way, that the last process in the cartesian MPI topology takes care of the remainder rows. Thus in some test scenarios (pair of number of processes and problem size N), there is a significant difference in the number of rows which are processed by the non-remainder processes and the remainder process. For example, in the case of $N = 1000$ and $n_p = 25$, where no remainders are required, there is a remarkable speed up compared to the case $N = 1000$ with $p = 24$.

It is clear that the speed up is greater for larger problem sizes. This is because there are more computations per communications for larger problem sizes than for smaller problem sizes. The efficiency graphs in figure 2 supports this fact. For a great number of processes and a small problem size, the efficiency is significantly lower than for larger problem sizes.

For n_p up to 10, the speed up for $N = 2000$ is almost ideal since almost linear.

For $n_p = 4$ and $N = 2000$ an efficiency of $E > 1$ can be observed. This means, that four processes within this execution are faster than one process operating only on a quarter of the grid. The reason for this might migt be likely due to some optimal conditions in this setup and less optimal setups for the same problem size with only one process, also overhead might be reduced.

For communication, non-blocking sends and receives are used without derived datatypes. The subgrids are implemented as row-major std::vector(subgrid_size) with northern and southern ghost layers.

In conclusion, the 1D decomposition depends on how the rows of the grids can be split between the processes and if unfavourable remainders exist. The remainder problem can be improved if the remainder rows are distributed most optimally over all processes. However, this increases the complexity of the code and the handling of the subgrid sizes.