3. Implementing a distributed reduction

The scaling of the optimized reduction is better, namely $\mathcal{O}(log(N))$ compared to $\mathcal{O}(N)$. Therefore, the optimized version is faster asymptotically. This was shown in the lecture and makes sense as the optimized version has $N/2$ ranks making a reduction in every time step, whereas the naive one only reduces from one rank to the master, while the other ranks are waiting. A second advantage is the load balancing. For the naive version we're freeing one rank at every time step, while we're freeing half the ranks in the optimized reduction. This means we can start a new computation earlier on more ranks. The argumentation for this is the same as in the point before.

4. MPI Bug Hunt

a) For multiple ranks we have a race condition. Both ranks open the same file and then write at the same time, making the output non-deterministic and not usable. This can be fixed by only letting one rank write, for example the root rank. We also have an issue with our loop index, as we're looping over $N + 1$ item while our array site is N, which can cause a segmentation fault.

b) We have a deadlock. Both ranks send first and wait for the other rank to receive. This can be resolved by reordering the send and receive instructions such that for example rank 0 is sending and then receiving while rank 1 is first receiving and then sending. Another solution would be to use a non-blocking point-to-point communication, for example `MPI_ISend`. Another bug is the type mismatch of the `MPI_Send` and the `MPI_Recv`. This can be fixed by changing the type in `MPI_Recv` to `MPI_DOUBLE`, as the value we're sending is a `double`.

c) The output for just one rank is probably the following: `error: ‘cout‘ was not declared in this scope; ...`. If we assume that the code is in the namespace `std`, then the expected output would be: `std[0] 0`. For two or more ranks the program will get stuck and not print anything, as the `MPI_Bcast` needs to be called by all ranks, so for more than one rank this code will not work. For just one rank it will work. Once we fix the issue from before, by moving the `MPI_Bcast` it works for any number $n \geq 1$. Another issue is that we're trying to use `MPI_Recv` to receive a broadcast. However, for collective communication we should use `MPI_Bcast` itself and then (possibly) put a barrier afterwards (if we need to synchronize the receiving of the broadcast). This is fixed by removing the `MPI_Recv`