Parallelle beregninger - TDT4200 - Problem Set 1

Ola Kristoffer Hoff

7. September 2022

1 Questions

1.1 What is the benefit of using MPI I/O in this program?

Since we have already distributed the workload so that each process works on a sub-grid of the entire solution, it is easy to make each process also write their sub-solution to file. This is possible because we can calculate where in the file each process needs to write its solution. There is also a lot of writing to I/O so the fact that we can parallelize this is a big advantage to avoid Amdahl's law which could be a factor in this program. It is also beneficial to think about what the program would need to do if the I/O was not parallelizable. Then the program would have to gather all the sub-solutions and put them all together, then it could write to file and become parallel again. So there would need to be a synchronization of the processes every time the program uses I/O. This I would imagine could become quite a bottleneck in the program, and hens is very beneficial to avoid.

1.2 What would happen if you use MPI_Ssend() (synchronous send) instead of MPI_Send() when communicating border values?

The main difference between **MPI_Send()** and **MPI_Ssend()** is the extra "s" which stands for *Synchronous*. It means that the send-call will not return until the receiving process has posted its receive-call. One can think of it like this: **MPI_Send()** is like sending a postcard, you write who it's for and post (send) it. When the counter part checks thier postbox (call receive is blocking, so they would stand and wait at the postbox until they got the expected postcard) they find the card. If three people; A, B and C, want to send post to each other in a ring (A to B, B to C and C to A), this is no problem they simply send their cards and go wait at the postbox until their post arrives (is received). This way everyone gets what they needed and this is what happens (roughly) in the program.
However, if we had used **MPI_Ssend()** it would be as if instead of sending the posting the postcard they deliverd it to the others house themself, and could not leave until the post was deliverd (recevied). If we try to do the same as above with A, B and C, we end up with A waiting a B's house, B at C's and C at A's. Then nobody delivers any post and hens can't go and get their own post either. This is called a *deadlock* where the program would frezze as the processes would wait eternally for each other.
This would not happen in my spesific program since I broke the ring because it makes no sense to share values from the east most side of the water with the west most side of the water, these have nothing to do with each other, and it is hens not needed to send data between them. I could have made the ring complete, but then i would have had to add checks too ensure that I ignored the data sharing between the to opposite sides, to get the correct solution, so I just didn't send any data at all.