Lab 6

Lab VI

Today we will practice concurrent programing by rewriting our serial code to one parallelized with MPI. The problem to be solved is the simple, linear advection with periodic boundary conditions. We will limit our attempts to the explicit version of the solver, but mind that implicit can be parallelized as well.

Domain decomposition

The usuall approach to parallelization for problems that "solve something" over a certain domain is to split this domain amongst processors. The assumption is that computational work is proportional to the number of unknows distributed over this domain. Those are usually associated with the computational mesh. Consequently the first step involves partitioning the mesh. Such partitioning should fulfill two main cryteria: * equally distribute the workload, * minimize resulting communications. Our mesh is rather simple, so partitioning will not be a problem.

Start

Start by adding the `MPI_Init();` and `MPI_Finalize();` to the begining and the end of your code. It is good to check if all components work and compile.

Partition your problem

Knowing the `size` of the problem and the number of processors, as well as the `rank` of the current process determine the local problem size and allocate memory accordingly. Then initiallize the problem, each processor works with its own chunk of data.

Parallel output

Redesign the `dump_solution` function so it can be used in parallel. You can use `MPI_Send` and `MPI_Recv` to synchronize the output.

Print current solution `u` and the `rank` so we can verify if all is OK using our favorite plotting program.

The explicit step

We need to modify one of the functions performing the explicit solution step. Since our problem is periodic and our partitioning "continous" the first process will need to communicate with the last and each process with the next one. This brings about a problem. We can either initialize the comunication such that the first communicates with the second, the second recives the message and than sends one to the next and so on. This does not seem like a good idea since processes will have to synchronize and than would wait for other processes to finish. To couter that we will apply the non-blocking communication. In the nonblocking communication the send/recive operations do not lock control, but mearly indicate that the buffer should be send to or recived from with the control imedietly returning to the caller. The price is the fact, that before clearing out the buffer or reading the data we will need to check if the comunication operation has completed. This is done with `MPI_Wait`, `MPI_Status` and `MPI_Request`.

We will discuss positioning the calls while coding this out.

Debugging

So the code is ready and compiles without errors. But is it working as expected? In general debuging a prallel code brings on a new dimension of problems as there is a number of new things that could be going wrong.