Short Course on Parallel Computing – Exercises

Peter Bastian, Olaf Ippisch, Sven Marnach, Jorrit Fahlke

Exercise 1 GNBP with MPI

In this exercise we will parallelise the n-body code using the Message Passing Interface (MPI).

1. As a first step, we will implement a ring shifting communcation scheme in the simple test example `mpi_ring.c`. Each process initializes a "buffer" (in this case a single integer) with some data (here simply the process rank). In each step of the ring communication, each process passes its current data to the next process in the ring, while it receives new data from the previous process in the ring. Because we are using blocking communication, we need "edge colouring" to implement this scheme. The processes with even rank first send and then receive data, while the processes with odd rank first receive and then send data (or vice versa).

   The code framework is already provided. Your task is to complete the code at the places marked with dots. Before the loop, the variables `rank` and `procs` must be initialised using `MPI_Comm_rank` and `MPI_Comm_size`, respectively.

   Inside the loop, the receive buffer (containing the information received during the previous loop iteration) is copied to the send buffer to pass this information to the next process. Then the communication is carried out using `MPI_Ssend` and `MPI_Recv`.

   Note that the required calls to `MPI_Init` and `MPI_Finalize` are already included in the example code.

   You can find information about the calling conventions of any MPI call in its man page (e.g. enter `man MPI_Comm_rank`). We always use `MPI_COMM_WORLD` as the communicator in these exercises.

   The program can be compiled with `make` and can be run with

   ```
   mpirun -np <number of processes> ./mpi_ring
   ```

   Keep the number of processes below 64 – you will run into resource limits otherwise.

2. Now we will apply this communcation scheme to the n-body code. To facilitate concentrating on the essential parts, the parallelisation of the support routines (generating the initial conditions, writing the VTK files) has already been prepared in the file `nbody_mpi.c` and supporting files. The current version of the code already works for a single process.

   In the MPI version of the code, each process only knows a fraction of the coordinates and masses. To keep things simple, the total number of points must be a multiple of the number of processes.

   You will have to perform the parallelisation of the routine `accelerate_mpi`. You need send and receive buffers which are able to hold coordinates of n points and n masses. To avoid copying the buffer in each loop iteration (as we did in the previous example with a single integer number in the buffer), you can use pointers to the two buffers and just switch the pointers in each iteration. In the first iteration the receive buffer is filled with the data of the local points. In the further iterations (but not in the first one), the communication is performed using the ring communication scheme from the first part of this exercise.

   You can test your code by spawning a small number of processes (≤ 8) on your local computer. You can compare the results of the computation with the vanilla version using the `fuzzy_diff` script from the OpenMP exercise. If you think your program works, you can run it on several computers – ask the supervisor for more information.

3. An alternative to the use of blocking communication and edge colouring is the use of non-blocking communication (with edge colouring not needed). Copy your program to the new name `nbody_mpi_async.c` and adapt the `Makefile` accordingly.

   Now at the beginning of each iteration (including the first one but excluding the last one), the asynchronous communication ist started using `MPI_Isend` and `MPI_Irecv`. While waiting for the data to arrive, each process does the computations on its current data. At the end of each iteration (again excluding the last one), we must wait for all data to arrive using `MPI_Waitall`.