Parallel Programming with MPI: Exercises

FICS Summer School 2010

CSC – Tieteen tietotekniikan keskus Oy
CSC – IT Center for Science Ltd.

General instructions

The exercises can be done on CSCs Louhi Cray supercomputer, in your own laptops, or in some other Linux based cluster. If you want to use your own laptop you need to have a Fortran compiler and an MPI library installed.

On Louhi you can either use your own Louhi account (if you have one), or alternatively one of the training accounts which will be provided to you during the first MPI lecture.

One should use a ssh client to connect to Louhi. On Windows there are several graphical ssh clients that you may use, while on Mac and Linux you can use the following command in the terminal:

```
ssh –X –l [username] louhi.csc.fi
```

The exercise materials can be found on the webpage, and under the `$WRKDIR` directory in Louhi when you use one of the training accounts.

Note that the training accounts will be reset and erased immediately after the course; please copy all the work you have done in them to a USB memory stick or to some remote server!

For editing files you can use e.g. `emacs`. Also other popular editors such as `vim` and nano are available.

Louhi User's Guide can be found at **http://www.csc.fi/english/pages/louhi_guide**

Compiling and running on Louhi

Compliling

You can compile your programs as follows:

```
Fortran:  ftn -o my_mpi_prog my_mpi_prog.f90
C:        cc -o my_mpi_prog my_mpi_prog.c
```

Running in a non-interactive mode

A program is run by creating a batch job script with a text editor which is then submitted to the queuing system. In the following example script a program is run using 16 MPI processes:

```
#!/bin/sh
#PBS -N test
#PBS -l walltime=00:15:00
#PBS -l mppwidth=16
cd $PBS_O_WORKDIR
aprun -n 16 ./my_mpi_prog
```

The job-script can be submitted to a queue as follows:

```
 qsub job_script
```

The output and possible error messages will be in files *test.oxxx* and *test.exxx,* where *xxx* is the system generated ID of the run.

Running in a interactive mode

For debugging purposes (and in this course) programs can be run interactively by launching aprun directly from the command line:

```
 aprun -n 4 ./my_mpi_prog
```

Compiling and running on a Linux PC or cluster

On Linux, there are usually commands **mpicc** and **mpif90** for building MPI programs. They can be used to compile a program as follows:

```
Fortran:  mpif90 -o my_mpi_prog my_mpi_prog.f90
C:        mpicc -o my_mpi_prog my_mpi_prog.c
```

The parallel program can be launched with the **mpirun** command:

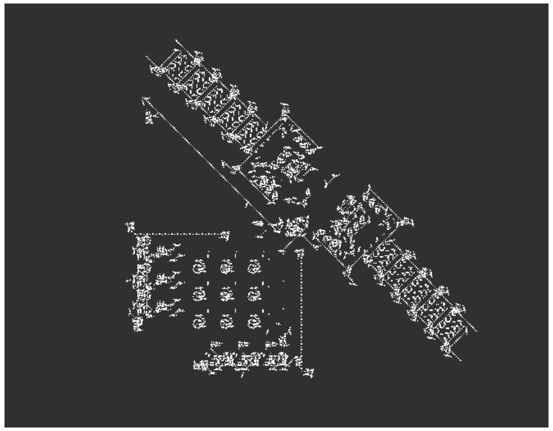
```
mpirun -np 4 ./my_mpi_prog
```

Exercises

The Game of Life (GoL) is a *cellular automaton* devised by John Horton Conway in 1970.

a) Study the GoL at
`http://en.wikipedia.org/wiki/Conway's_Game_of_Life`

In this exercise, a straightforward MPI implementation of the Game of Life is discussed. A working serial version of GoL is given in `exercises/gameoflife_serial.f90`. This program draws the board into file `life_nn.pbm`, where nn is the iteration. The board is printed after every few iterations, that is adjustable You may view the image with the `xview` command, e.g. "`xview -zoom 400 life_100.pbm`"



b) Your mission is now to parallelize the GoL with MPI, by dividing the board in columns and assigning one column to one task. The tasks are able to update the board independently everywhere else than on the column boundaries - there communication of a single column with the nearest neighbor (the board is periodic, so the first column of the board is 'connected' to the last column) is needed. Make all the MPI tasks print their own parts of the board on different files, e.g. `life_nn_mm.pbm`, where nn is the iteration and mm is the rank ID. You may start directly from the serial version, or by inserting the proper MPI routines into a skeleton code available at `exercises/gameoflife_mpi.f90`

c) Perform the column exchange communication with non-blocking routines. Try to overlap the communication with the interior board update.

d) Modify the output such that the full board is printed in a single file. You may do this by gathering the full board in a single MPI rank, which prints it out.