Spring 2015: Advanced Topics in Numerical Analysis: High Performance Computing Assignment 1 (due March. 9, 2015)

1. **MPI ring communication.** Write a distributed memory program that sends an integer in a ring starting from process 0 to 1 to 2 (and so on). The last process sends the message back to process 0.

   - Allow for a command line parameter N that specifies how often the message is sent around the ring.
   - Start with sending the integer 0 and let every process add its rank to the integer before it is being sent again. Use the result after N loops to check if all processors have properly added their contribution each time they received and sent the message.
   - Time your program for a larger N and estimate the latency on your system (i.e., the time used for each communication). If possible, try to test your communication ring on more than one machine such that communication must go through the network. Note that if you use MPI on a single processor with multiple cores, the available memory is logically distributed, but messages are not actually sent through a network.[1]
   - Hand in your solution using the filename `int_ring.c`.
   - Modify your code such that instead of a single integer being sent in a ring, you communicate a large array of about 2MByte in a ring. Time the communication and use these timings to estimate the bandwidth of your system (i.e., the amount of data that can be communicated per second).

2. **Distributed memory parallel Jacobi smoother.** Use MPI to write a parallel version of the Jacobi smoother from Homework #0. In particular:

   - Distribute each vector uniformly amongst the available processor cores p. If N is not a multiple of p, exit with an error message.
   - Note that the ith process only has to allocate memory for the part $\boldsymbol{u}^i$ of the vector $\boldsymbol{u}$ it is working on, plus the left and right point it requires for the stencil computation. The vector $\boldsymbol{u}^i$ is thus of length $n = N/p + 2$.
   - Use `MPI_Send` and `MPI_Recv` to, after each Jacobi step, communicate the left and right values of $\boldsymbol{u}^i$ to the processes that need these values for the next Jacobi step.
   - Verify that the result is independent of p. By trying out different p while leaving N fixed (and large), study the strong scaling of your program. Use a fixed number of Jacobi iterations (say, 10) for this comparison.
   - Hand in your solution using the filename `jacobi-mpi.c`.
   - Why is a parallel version of the Gauss-Seidel smoother be significantly more difficult?

[1] It depends on the implementation of MPI how it handles sending/receiving of data that is stored in the same physical memory.

Handing in your homework

Please create a Git repository on either Github or Bitbucket. If you choose your repository to be private, please add me to the repo (my user name on both is `georgst`). The repository should contain the files `jacobi-mpi.c` and `int_ring.c` (no need to hand in the code for the ring problem for the 2MB message–just report your observations), as well as a Makefile. My MPI compiler is simply called `mpicc`[2]. You can either email me the location of your repo and add a short `.txt` of LaTeX file to the repo that answers the questions from this assignment, or you hand in a sheet with the answers that also gives me the location of your repo. To check if your code runs, I will type the following commands[3]:

```
git clone YOURPATH/YOURREPO.git
cd YOURREPO
make
mpirun -np 10 int_ring 1000
mpirun -np 10 jacobi-mpi 10000 10
```

For the Jacobi problem, I assume that the first input parameter is N, the number of grid points, and the second parameter is the (maximum) number of iterations.

[2]If you want to change the name of your compiler on your machine, you can create an alias for it in your `.bashrc` or `.cshrc` (or whatever shell you use). On my Mac, for instance, I use: `alias mpicc='mpicc-openmpi-mp'`

[3]I want to partially automate these steps, so please make sure that this works; in particular, call the executables your Makefile creates accordingly.