Assignment 4

Introduction to Parallel Programming

due **Thursday 26 October 2017, 18:00** (hard deadline)

Instructions

As in the previous assignments, you will need to register in a group on Studentportalen. The group can be different than your previous ones, but, once again, we suggest that you find another student to work together and form a group. In case of trouble, please contact Stephan (stephan.brandauer@it.uu.se).

Submission checklist:

- Submissions must clearly show your name(s).
- Submit a **single** PDF report, as well as all source code related to the exercises in Studentportalen.
- Solutions must be in C or C++ with MPI.
- All source code must compile and run on the **Linux lab machines that have MPI installed**. For the list of these machines, please see instructions below.
- **Provide instructions for compilation and running, preferably by including Makefile(s).**
- No source code modifications should be required for reproducing your results.
- Your report must describe relevant details of your solutions and report their performance.

In case you do not reach a working solution, describe the main challenges and proposals to address them. Also, note that some of the exercises ask you to do benchmarking on the Lab machines and this requires 1) using the machines at a time when they are lightly loaded, and 2) taking multiple measurements to see if there is any variation. Thus, it's not a good idea to leave this part of the assignment for too close to the deadline!

Exercise 0: "Hello, MPI World!" (0 points; no need to submit)

The slides of the MPI lecture contained two versions of a "Hello, World!" program using MPI as well as a program that calculated π using MPI broadcast/reduce.

a) Open an editor and write or copy-paste the *second* "Hello, World!" program (the one printing out the size and rank), in a file named `hello_mpi.c`. Now it's time to familiarize yourself on how you can use MPI programs on the lab machines. Some instructions to do this appear in Figure 1, so follow them step by step.

b) After you finish with the above task, do something similar with the program that calculates π. (You may need to include some additional header file(s) and comment out some variable(s), e.g., `done`.)

c) The Web contains a myriad of tutorials for MPI if you need more information to do the exercise that follows. Enjoy reading!

Start working on the server `gullviva.it.uu.se`. You will be using the servers `tussilago.it.uu.se` and `vitsippa.it.uu.se` as worker nodes.

1. Make sure `ssh` is configured to not require passwords. You can check this with the following command:

   ```
   ssh your_username@gullviva.it.uu.se.
   ```

2. Make sure that you can log into `tussilago` and `vitsippa` from `gullviva` without having to specify a password by following these instructions: http://linuxproblem.org/art_9.html.

3. Create the file "`hosts`", with the following three lines as content:

   ```
   gullviva.it.uu.se
   tussilago.it.uu.se
   vitsippa.it.uu.se
   ```

4. Compile your "Hello, World!" program with the following command:

   ```
   mpicc hello_mpi.c -o mpitest
   ```

5. Find out information about the MPI function `MPI_Get_processor_name()` and modify the "Hello, World!" program appropriately so that it also prints the name of the server each "processor" of your MPI program is running on as shown below. Compile your program again after you have modified it.

6. Run your program using the command

   ```
   mpiexec --hostfile hosts ./mpitest
   ```

   If all goes well, you should be seeing output like:

   ```
   size=96, rank=1 (gullviva.it.uu.se)
   size=96, rank=2 (gullviva.it.uu.se)
   size=96, rank=3 (gullviva.it.uu.se)
   ...
   size=96, rank=64 (vitsippa.it.uu.se)
   size=96, rank=33 (tussilago.it.uu.se)
   size=96, rank=65 (vitsippa.it.uu.se)
   size=96, rank=34 (tussilago.it.uu.se)
   size=96, rank=66 (vitsippa.it.uu.se)
   size=96, rank=35 (tussilago.it.uu.se)
   ...
   size=96, rank=75 (vitsippa.it.uu.se)
   size=96, rank=79 (vitsippa.it.uu.se)
   size=96, rank=70 (vitsippa.it.uu.se)
   ```

   In case you see error messages like `...: command not found`, try executing this line:

   ```
   export PATH=/usr/lib64/openmpi/bin/:${PATH}
   ```

   Alternatively, add it at the end of your `~/.bashrc` file.

Figure 1: Instructions for the MPI lab.

Exercise 1: Sieve of Eratosthenes (2 + 1 + 3 = 6 points in total)

The previous assignments asked you to implement a parallel version of the Sieve of Eratosthenes algorithm for finding prime numbers using Posix threads and OpenMP.
This exercise asks you to use MPI instead of shared-memory parallelism constructs for the parallelization of your solution. Similar constraints as in Assignment 3 apply: If you choose to work in the same (one or two person) group as before, you need to use the program of your previous submission(s) as a basis for this one. If you decide to form a *new* group with another student for this assignment, you can choose *one* of your previous submissions as basis (state which one you used in your report). If you have *not* submitted a program for this exercise before, you can of course write an MPI solution from scratch.
You need to provide and/or measure the following:

1. An MPI program using `MPI_Send()` and `MPI_Recv()` measuring its performance on *one* server.
2. The same program as above measuring its performance on the *three* servers mentioned in Figure 1.
3. An MPI program using `MPI_Bcast()` and `MPI_Reduce()` measuring its performance on three servers.

Make sure you use an N which is large enough for the parallelization to make sense.
Hint (and Warning): You can find various tutorials on the web on how to best parallelize the Sieve of Eratosthenes using MPI and also programs that do this. Your solution need not be super sophisticated or optimized, and it should also not be very similar to code "out there".
Besides your code, you need to provide a short section in your report that explains how you modified your solution and reports the speedup curves you got. Was the MPI version easier or more difficult to write? Are the speedups you get the same better or worse (and why)? Refer to previous assignments for more information on how to benchmark your program.

Good luck!