# Practical Session with MPI

21. Okt. 2008

## 1 Content of src/

The src/ directory contains the following programs:

- "hello-mpi.c": hello World!
- "ping-pong.c": ping-pong with MPI.
- "pi.c": sequential program that computes an approximation of $\pi$.
- "mslave.c" : a very basic Master/slave program with MPI.

In Linux, all the usual C, shell and MPI commands should be documented through the command `man`. For instance, type in a shell: `man gcc` to get all the options to use the GNU C compiler.

## 2 Exercises

**1. Running a MPI program** Compile and run the "hello world!" program. The command to compile a MPI program is `mpicc` (it is a simple wrapper for gcc, so that the normal compilation flags apply. For instance, try `mpicc -o hallo -g hello-mpi.c`.) In order to run the MPI program, use `mpirun -np 4 ./hallo`, where 4 is the number of processes that you want to run.

**2. Ping-PONG** . The "ping-pong" program actually only is a "ping" program: a given process sends a message to another one. Adapt it so that:

1. the process 0 chooses a destinatory process randomly;
2. 0 sends a message containing 10 integers to the destinatory;
3. the receiver computes the sum of the 10 integers and sends it back to 0;
4. 0 receives the sum and prints it to the screen.

In C, you can use for instance `nrand48` to generate a random int number. Do not forget to initialize the array of 10 numbers before sending it. To print the value of an integer `x` in C, use something like `printf("x is set to: %d\n, x);`.

In your program, $p-2$ processes will probably be blocked in a Receive from 0, while 0 does not send anything to them. Try to work out a solution to avoid these blocked processes.

**3. Computing $\pi$.** In the *pi* program, there is a main loop which iteratively refines the $\pi$ approximation. Parallelize it with MPI, helping yourself with the naive Master/Slave program also provided: a master process (for instance the process number 0) should send messages to slaves processes. Each message indicates to a slave that it must compute $K$ iterations of the loop, starting at step $i$. $i$ is an integer that must be sent into the message, $K$ is a predefined constant (for instance, $K = 1000$). Thus, each task that a slave will run is a bunch of $K$ iterations. The slave must sum up all the contributions of these iterations and send this partial result back to the master, who must then add it to its whole global approximation of $\pi$, and then send a new task to the slave that has finished.

Take care about the following points:

- when the master process has sent all its tasks to the slaves, it still has to wait for the results of the last running tasks.
- Each slave will execute a loop to wait for a task from the master. Since it does not know initially how many tasks it will have to run, the slave must run an infinite loop. It will be up to the master to send a final message to each slave in order to tell him that there is no more task to run, and that he can exit.

**4. Communication in a ring.** This exercise must be resolved and sent to `nicolas@inf.ufrgs.br` and `mrpritt@inf.ufrgs.br` until Oct., 30th, together with the measurements of the performance of the network and your conclusions about it.

All the $p$ processes must communicate using a logical ring and a token. The token is a variable that contains a value to be broadcast among the processors. The process of rank 0 sets its value, and then sends the token to its neighbor process ranked 1. Each process of rank $r = 1...p-1$ must wait for an incoming message from its neighbor of rank $r-1$, read the value of the token and send it to its neighbor process of rank $i+1$. By convention, the neighbors of the process of rank $p-1$ are $p-2$ and $p == 0$. When the process 0 gets the token back, the broadcast is over.

Notice that this ring communication is a generalization of the "ping-pong" scheme.

You can use this algorithm to evaluate the network performance: how many messages will be exchanged for one whole broadcast? How can you use the size of the token and the number of messages to evaluate the latency and the throughput of the communication links between the processors?

You will need to measure the time it takes for a program (or part of it) to execute. You can use for instance `MPI_Wtime` this way:

```
double starttime, endtime, time_used;
/*... Whatever ... */
starttime=MPI_Wtime();
/* Sequence of instructions whose timing you want to get... */
endtime=MPI_Wtime();
elapsed_time = endtime-starttime; /* seconds */
```