CPDS Laboratory Hands–On
MPI: Brief tutorial

J.R Herrero

Fall 2023

Contents

Examples 1

Message Passing Interface (MPI)

1.1 Learning MPI at boada

We post all necessary files to do each hands–on and each assignment in `/scratch/nas/1/cpds0/sessions`. For the session today, you need to extract the files from file `HO_MPI.tar.gz` **from** that location by uncompressing it **into** your home directory in `boada`. From the the root of your home directory unpack the files with the following command line: `"tar -zxvf /scratch/nas/1/cpds0/sessions/HO_MPI.tar.gz"`.

Recall from *Assignment 0* that, in order to set up all environment variables, you have to process the `environment.bash` file now available in your home directory with `"source ~/environment.bash"`. As stated in that assignment, it is convenient to do this every time you login in the account or open a new console window and, therefore, we recommend to have this command line in the `.bashrc` file in your home directory, a file that is executed every time a new session is initiated. If you have unpacked the a0.tar.gz files in your home directory then this file has already been added automatically to your home directory and it will be used from your next connection to `boada`.

In case you need to transfer files from `boada` to your local machine (laptop or desktop in laboratory room), or viceversa, you have to use the secure copy `scp` command. For example if you type the following command `"scp cpds1XYZ@boada.ac.upc.edu:a0/MPI/pi/pi_seq.c ."` in your local machine you will be copying the source file `pi_seq.c` located in directory `a0/MPI/pi` of your home directory in `boada` to the current directory, represented with the `"."`, in the local machine, with the same name. Plese, review *Assignment 0* in case you have doubts on how to use `boada`.

1.2 Programming, Compiling and Executing MPI programs

1.2.1 Learning `MPI`

In this section we are going to use `MPI`, the standard for parallel programming using distributed-memory systems, to express parallelism in the C programming language. In Assignment 0 you can find some codes computing digits of number Pi in `a0/MPI/pi`. We used those codes in order to observe the strong scalability of that code using `MPI` processes. However, at that stage we skipped going through the code since the purpose was motivating you to learn about parallel programming in view of the reduction in execution time it can achieve (in some cases). Contrary to Assignment 0, in this part of the course we need to understand how to develop a code using `MPI`. We introduce `MPI` in the videos available at Atenea. Please, watch them first and do the associated questionnaires. Next, you will find a set of examples in directory `HO_MPI/Tutorial`. Use them to test different functionalities offered by MPI. They come from `https://www.jics.utk.edu/mpi-tutorial` at the University of Tennessee. You can find explanations there. In most subdirectories there is a file with name ended in `.c.soln` with the solution so that you can check your solution against a good one. You can disregard anything related to PBS, which is another queueing system different from SLURM, the one used at `boada`. In summary, please whatch the videos on `MPI` and try to understand the code and behaviour of the programs in the Tutorial, at least for the following programs: `hello1, hello2, pical, pingpong, deadlock, deadlockf, collectives`.

1.2.2 Write your own `MPI` code

Finally, test your understanding by creating an `MPI` program of your own which includes calls to several `MPI` primitives. Here's your opportunity to be creative! It's not necessary that the code is really useful, but the code should include some message exchanges with both 1) point to point primitives and 2) at least one collective communication primitive.

For the latter, notice that all processes have to call a collective primitive in the very same way in the program; and the way by which we can specify which of the processes has to behave differently is through one of the parameters. Oftentimes such process is called the *master* or *root* process in the literature.

In the introductory video and in multiple Makefiles within the `MPI` subtree of directories provided you can see how to compile C programs extended with `MPI` with `mpicc`. But, if you prefer to code the `MPI` program of your own using C++ you can compile it with `mpic++`. If you prefer Fortran then you can use `mpif77`, `mpiF90` or `mpifort` depending on the flavor of Fortran.

For instance, if my `MPI` program is `hello2.c`, we can compile with "`mpicc -o hello2 hello2.c`". Since the code you will produce is likely to run in a very short time you can execute it interactively with `mpirun`. For instance, we can run it with 4 processes with "`mpirun -np 4 hello2`".

Note: if your code involves a lot of CPU time then you have to create a shell script and launch it to execution in the queuing system with the `sbatch` command. If that was the case, you can get inspiration from the multiple `submit-xxxx.sh` files within `Assignment 0`. However, this is NOT the goal and we neither expect you to develop a complex code here nor one that implies a high computational cost.

Please, test your code running it with several processes.

Examples 2

Additional examples

We have compiled a few more examples which can be useful for learning additional `MPI` primitives. You might be interested in learning from them, either at this point, or while solving the *assignments* in this course. Check directory `HO_MPI/OtherExamples/` for those examples. Compile them all by executing `./cl.sh`. Then, study each code and the output of its execution under `mpirun`. Try execution with different number of processes whenever possible.