Uppsala University
Information Technology
Jarmo Rantakokko

Programming of parallel computers
MPI Computer Lab
2013-01-16

MPI Exercises

Getting started

In the lab you will need a number of test programs. The source files for these programs are located in Studentportalen under File area/Labs. Copy the program files into your directory and unpack them with `tar xvf MPIfiles.tar`. This will create to subdirectories, MPIbasic and MPIpde, with the files. Go to the subdirectory MPIbasic.

1 Compile and run MPI programs

Study the `Makefile` to see how to compile MPI programs. Notice that we use `mpicc` and link with the MPI library `-lmpi`. The runs are then handled with the command `mpirun -np x a.out` requesting x processes to run program a.out.

- Compile the program hello, `make hello`. Run the program on one process, `mpirun -np 1 ./hello`. Study the program source file and compare with the output. Re-run the program on more than one processor.

- Modify the program so that each processor prints its number (rank) in addition to the greetings. Also make the processor with rank zero to print the total number of processors that take part in the current execution of the program. For this task you will need to call the MPI functions `MPI_Comm_size` and `MPI_Comm_rank`. For correct syntax and behavior see `http://www.mpi-forum.org/docs/mpi-11-html/node182.html`. Re-run the program several times, is there a predefined order of the output?

2 Point-to-point communication

In the program `exchange` two processors exchange the variable a between each other, and save it as b using point-to-point communication.

- Compile the program `make exchange` and run with two processes, `mpirun -np 2 ./exchange`. Study the source file and compare with the output of the program. Modify the program to use non-blocking communication with `MPI_Isend, MPI_Irecv,` and `MPI_Wait`. Make sure that your program is correct by consulting the instructor. What are the advantages of using non-blocking communication?

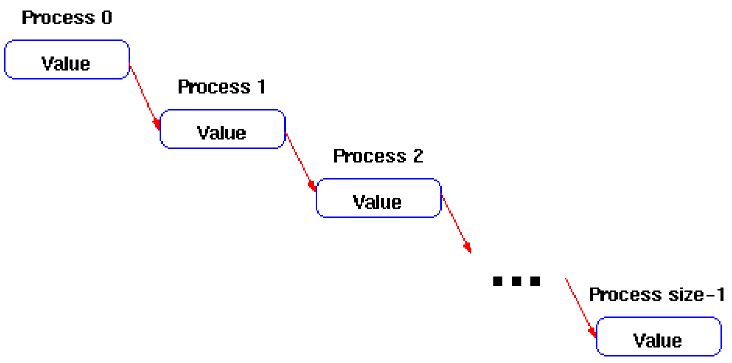
- Study the program file `pingpong`. The code uses a synchronous `send` and a blocking `receive` to send a message between two processors back and forth. Similarly to the other cases, build up the executable by `make pingpong`. Run the code on two processors and estimate the latency (start-up time) and the maximal bandwidth (transfer rate) for point-to-point communication. Use the Matlab file pingpong.m to compute the estimations.

3 Collective communication, global data

Here we will study the case where we distribute data from one processor to all others. In the program `onetoall` we solve this by sending data from processor 0 to the others one at a time.

- Compile and run the program *onetoall*. Study the source file and compare with the output.

- One problem with the solution above is that processor 0 is more loaded than the others. Modify the program to a *pass-on* sequence as in Figure 1 to get a more fair load between the processors.



Figur 1: Global data sharing using a pass-on sequence.

- **Challenge** (optional): In the *pass-on* sequence we have a serialized communication flow. A more efficient way to globally share data is to use a *fan-out* sequence, i.e., in the first step processor 0 sends to processor 1, in the second step processor 0 sends to processor 2 while processor 1 sends to processor 3, in the third step all four processors sends to processors 5-7, etc. Modify your pass-on program to implement the fan-out sequence with point-to-point communication.

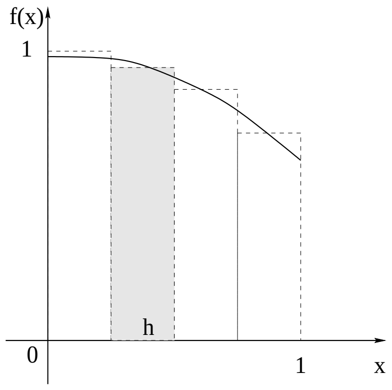
- MPI provides a set of global communication operations, called *collective communication*, using efficient implementations optimized for the underlying communication network. For one-to-all communication we have `MPI_Bcast`. Modify the program `onetoall` to use `MPI_Bcast` instead of point-to-point communication. Compile, run, and verify its correctness. What other collective communication operations are provided in MPI?

4 Reduction, global operations

The program `pi` computes π in parallel using numerical integration. A numerical way to compute π is the following:

$$\int_0^1 \frac{4}{1+x^2}dx = [4\,arctan(x)]_0^1 = \pi$$

If we use the midpoint rule, we can compute the above integral as follows (see Figure 2):



Figur 2: Numerical integration with the midpoint rule.

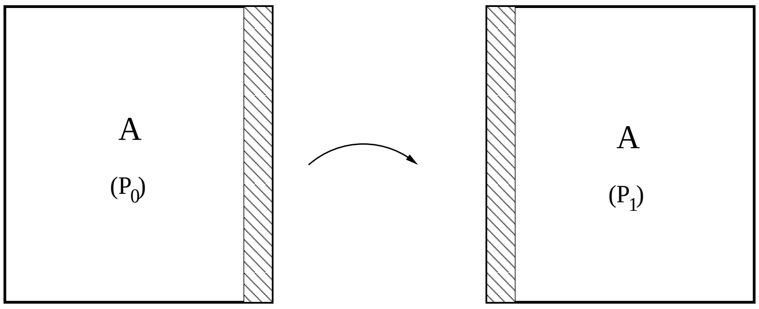
$$\int_0^1 \frac{4}{1+x^2}dx = h\sum_{i=1}^{n}\frac{4}{1+x_i^2} = h\sum_{i=1}^{n}\frac{4}{1+((i-1/2)h)^2}.$$

The trivial parallel implementation of the latter formula is that if we have p processors available, we slice the sum into p pieces, attach one interval to each of them to compute a partial sum, and then collect the local sums into one processor, which will know the answer. Complete the code using the global reduction operation, `MPI_Reduce`, to collect the local sums to a final result. (For usage see `http://www.mpi-forum.org/docs/mpi-11-html/node182.html`.) Verify correctness running on different numbers of processors. Why does the value of pi differ and is this acceptable?

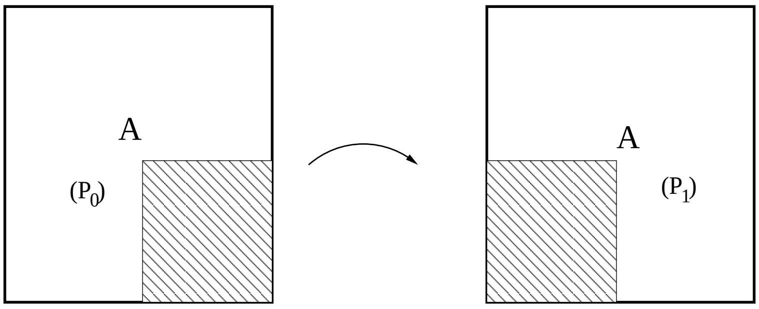
5 Derived datatypes

Suppose that we want to communicate one column of a 2D array and that we store data row-wise in a 1D data structure, see Figure 3. The arguments to `MPI_Send` and `MPI_Recv` specifying the data are its type, length and starting address. This requires data in one message to be stored contiguously in memory which causes a problem in our case.

We can then either send the elements one by one (very inefficient!) or copy the column to a temporary array, send the temporary array, receive data in another temporary array and finally unpack the temporary array to the corresponding column. However, MPI provides a more convenient way by the use of derived datatypes. Compile, run and study the program `datatypes`. Make sure that you understand the usage of `MPI_Type_vector`. Modify the program to communicate a block corresponding to a quarter of the 2D array as seen in Figure 4.



Figur 3: Communicate a column from P_0 to P_1, assume data stored row-wise in a 1D array.



Figur 4: Communicate a block from P_0 to P_1.

6 Communicators

MPI uses communicators to separate messages into different communication channels (messages are not mixed between two communicators) and to define collective operations for a subset of the processors. Moreover, it provides a convenient way to name and order the processes, e.g., into a cartesian grid. The program `communicators` organizes the processors into a 2D cartesian grid and creates sub-communicators for the rows. What is the value of `mydata` on respective processor after the broadcast operation? Compile and run the program to verify your answer. Modify the program to create sub-communicators for the columns instead. What is value of `mydata` now?

7 Putting it all together, an advanced example

Consider the hyperbolic Partial Differential Equation

$$\begin{aligned} u_t + u_x + u_y &= f(x,y), \quad 0 \le x \le 1,\ 0 \le y \le 1 \\ u(t,0,y) &= h(y-2t) + u_p(0,y), \quad 0 \le y \le 1 \\ u(t,x,0) &= h(x-2t) + u_p(x,0), \quad 0 \le x \le 1 \\ u(0,x,y) &= h(x+y) + u_p(x,y), \quad 0 \le x \le 1,\ 0 \le y \le 1 \end{aligned}$$

where

$$
\begin{aligned}
f(x,y) &= 2e^{x+y} + 3x^2 + 6y^2 + \sin\left(\frac{x+y}{2}\right) + \cos\left(\frac{x+y}{2}\right) \\
u_p(x,y) &= e^{x+y} + x^3 + 2y^3 + \sin\left(\frac{x+y}{2}\right) - \cos\left(\frac{x+y}{2}\right) \\
h(z) &= \sin(2\pi z)
\end{aligned}
$$

This equation has the solution

$$u(t,x,y) = h(x+y-2t) + u_p(x,y) \qquad \text{(i)}$$

We can also solve the PDE numerically with for example the Leap-Frog scheme

$$u_{ij}^{n+1} = u_{ij}^{n-1} + 2\Delta t(f_{ij} - D_{0x}u_{ij}^n - D_{0y}u_{ij}^n)$$

where

$$D_{0x}u_{ij}^n = \frac{u_{i+1,j}^n - u_{i-1,j}^n}{2\Delta x}, \quad D_{0y}u_{ij}^n = \frac{u_{i,j+1}^n - u_{i,j-1}^n}{2\Delta y}$$

But now we need extra numerical boundary conditions on the lines $x = 1$ and $y = 1$. Here we can extrapolate the solution from the inner to the boundaries but for the simplicity we choose to use the analytic solution (i).

The solver includes the following routines and are located in the subdirectory MPIpde that was created in the beginning of the lab (Getting started):

`wave.c`	(main program)
`wave.h`	(header file, function declarations)
`bound.c`	(boundary conditions)
`diffop.c`	(differential operators)
`initcomm.c`	(communication setup)
`residual.c`	(error norm)
`force.c`	(forcing functions f, u_p, h)
`nodes`	(server names)
`Makefile`	(makefile)

The code is parallelized in two dimensions and each processor is responsible for the calculations on a block: $x_1^{(p)} \le x \le x_2^{(p)}$, $y_1^{(p)} \le y \le y_2^{(p)}$. The processors are organized in a 2D cartesian topology to match the data decomposition. Communication is needed in the D_{0x} operation, D_{0y} operation, and in the residual computation. The communication is local neighbor to neighbor and is repeated in the same pattern in each timestep. In the y-direction the data is not contiguous and derived datatypes are used. To lower some of the communication initialization overheads *persistent communication* is used. Moreover, the communication is overlapped with the computations to minimize the synchronization overheads.

Compile and run the program using different numbers of processors, note the timings for each run. Plot the *speedup*, $S_p = T_1/T_p$, where T_1 is the time to run the program on one processor and T_p the time to run the program on p processors. Is the scaling linear, can you explain the results, is it what you expected?

To run on more than one node (more than 8 cores) you need pass a machine file as argument to mpirun. Use the command `mpirun -hostfile nodes -mca plm_rsh_agent rsh -np 20 a.out` to run 20 processes distributed to the machines specified in the file `nodes`. Please, be careful not to run on all cores on all machines, limit your runs to 64 processes, otherwise other students will suffer from slow response time on the system. (Obvious misuse can lead that you can be suspended from the computer system.)

8 Report

The laboration is a part of the examination. To pass you need to either attend the lab and work actively on the tasks or write a short informal report summarizing your results and answers together with your source code for the different tasks.