|epcc|

Advanced Message-Passing Programming Exercises
MPI Performance

David Henty

1 Introduction

The purpose of this exercise is to investigate the performance of basic point-to-point MPI operations. You are given a simple ping-pong code that exchanges messages of increasing size between two MPI ranks, and prints out the average time taken and the bandwidth over a large number of repetitions. With this code you can investigate how the following factors affect the performance:

- different underlying protocols for different message sizes;
- different communication hardware (e.g. over the inter-node network or intra-node memory copies);
- different MPI send modes (e.g. synchronous vs. buffered);
- different MPI datatypes (e.g. contiguous or strided data);
- different NUMA regions within a node.

2 Compiling and Running

The code is contained in `AMPP-pingpong.tar` on the course web pages. You should be able to compile it using `make`, and submit the supplied Slurm scripts unchanged. As distributed, the Makefile is set up for Cirrus but there are comments on how to compile on ARCHER2.

On Cirrus, you will need to load some non-default modules:

```
[user@cirrus ~]$ module load mpt
[user@cirrus ~]$ module load intel-compilers-19
[user@cirrus ~]$ module load gnuplot
```

By default, the code runs on two processes each placed on the same node (so communications will **not** take place over the network), and benchmarks standard and synchronous sends. Note that **the program also prints out the exact location of the two processes**.

For each mode, two ".plot" files are written containing the times and the bandwidths as a function of message size. The results for the two different modes can be compared using:

```
gnuplot -persist plot_time.gp
gnuplot -persist plot_bandwidth.gp
```

Check that you understand the general form of the graphs before proceeding.

3 Experiments

The supplied gnuplot ".gp" files compare the results for standard and synchronous modes. If you want to compare different modes, or more than two modes, you will have to edit the gnuplot files – the format should be self-explanatory. Also note that the code overwrites the ".plot" files so you will need to make copies after each run if you want to keep a record of the results (e.g. when changing the placement of processes on the cores or nodes).

If you run on more than two processes, the program sends messages between the first (rank zero) and last ranks with all the other processes remaining idle. This is useful when altering the assignment of processes to cores.

By varying `nodes` and `tasks-per-node` in the Slurm script you should have complete control of the placement of the first and last processes.

After having run the default setup – two processes on cores 0 and 1 – you should experiment with the following:

1. Send between different nodes, e.g. run with a total of two processes with one process on each of two nodes.
2. Communication between two processes widely separated on the same node (e.g. `nodes=1` and `tasks-per-node=36`).
3. Can you explain any variations in the latency and bandwidth as you change the process locations?
4. Investigate the performance of buffered or strided sends.