|epcc|

MPI-2 single-sided communication on ARCHER

September 6, 2017

1 Introduction

The purpose of this exercise is to investigate the single-sided functionality of MPI-2. You will start with a working MPI parallelised image processing code and replace the point-to-point functions for halo-exchange with MPI-2 single-sided operations.

2 FENCE and PUT

Your first single-sided version will use `MPI_WIN_FENCE` for synchronisation and `MPI_PUT` for data movement.

- first, add window creation and tidy-up function calls,
  use `MPI_WIN_CREATE` to create a window
  and `MPI_WIN_FREE` to destroy the window.
  Test your program still works and gets the right answer.
- next, add synchronisation function calls,
  use `MPI_WIN_FENCE` before your main loop
  and `MPI_WIN_FENCE` inside your main loop.
  Test your program still works and gets the right answer.
- finally replace the non-blocking point-to-point function calls with `MPI_PUT` function calls.
  Remember to remove the calls to `MPI_WAIT` or `MPI_WAIT_ALL`.
  Test your program still works and gets the right answer.

Now, for various numbers of processes, benchmark your new single-sided program with images of different sizes.

Is your new code faster or slower than the point-to-point version you started with?
Does your new code strong-scale (same input image, more processes) well or badly?
Does your new code weak-scale (bigger input image, more processes) well or badly?

3 FENCE and GET

Make a new version of your image-processing code that uses `MPI_GET` instead of `MPI_PUT`. You should be able to re-use the window creation, synchronisation using `MPI_WIN_FENCE`, and tidy-up code without any changes.

Is this new code faster or slower than the version using `MPI_PUT`?
Does your new code strong-scale (same input image, more processes) well or badly?
Does your new code weak-scale (bigger input image, more processes) well or badly?