HPC Practice Lab 04: MPI One-sided Communication

- Deadline for submission: 27.11.2018, 23:59 -

In this exercise we will create a renderer for the Mandelbrot set, which is an iconic example of a fractal. Refer to https://en.wikipedia.org/wiki/Mandelbrot_set for more information. The purpose of this exercise is to get practical experience with one-sided MPI communication.

Overview

We will divide the area to be rendered among multiple MPI processes. Each process will perform its part of the calculation and write its results directly into the memory of process 0 using RMA operations. At the end, process 0 will output the result as a .ppm image file, which then can be viewed.

A code template is provided with this exercise. While you are working on Noctua, you can copy it directly into your current working directory:

```
cp $PC2DATA/hpc-lco-plessl/lab04/mandelbrot.c .
```

The following tasks will let you complete this template step by step. Have a look at the comments in the template as well!

Task 0: Understand the code

Read the code template. Try to understand what it is doing and what parts are still missing.

Questions you should be able to answer:

- How do I run the code?
- What are the command line arguments?
- How many processes do we need to execute the code?

Task 1: Implement data transfer

Transfer the pixels calculated in `HandleBlock()` to the memory exposed by Process 0 using `MPI_Put`. The positions to write to can be found as fields of the `block` argument. If the data is properly transferred, a triangular shape should appear as the output.

Run the program locally with `mpirun` or using `srun`. When using `srun`, make sure to allocate CPU cores instead of entire nodes by using the argument `-n` instead of `-N`.

Task 2: Implement logic

Implement the actual logic for determining whether a point is in the Mandelbrot set in `checkMandelbrot()`. Read the definition of the Mandelbrot set in the linked Wikipedia article. The real and imaginary components of the point $c = \mathrm{Re}(c) + i\,\mathrm{Im}(c)$ are given to the function as `real` and `imag`.

The maximum number of iterations to perform is given to the function as `cutoff`. Perform the iteration up to as many times as `cutoff` states. If the absolute value is ever greater than 2, the series will diverge for sure and you can conclude that the point is not in the set and immediately return. Otherwise, you assume the point is in the set.

Note: The function should return the iteration count n for which $|z_{n+1}| > 2$, or `cutoff` if the point is in the set.

Task 3: Distribute the load

As of now, your program should run and render the Mandelbrot set, but only use a single process that does all the work. Now divide up the square plane into multiple blocks, and assign them to the different processes. It is noteworthy that some blocks require way more work than others (those inside the set will have each pixel run until `cutoff`). This is likely to cause load imbalance. To combat that, create more than one block per process and assign blocks to processes in a way that distributes the load.

Expected Results and Suggestions for Experiments

- Scale up the number of processes. You should see a speedup.
- Try passing a higher number of iterations on the command line.
- With the higher iterations rendering a zoomed-in version of an interesting part of the fractal.

Submission

As usual, submit your code and a script that performs the following tasks:

- Setting up the necessary environment,
- compiling your code, and
- launching your code with `mpirun` or `srun`.

Happy coding! ☺

Helpful resources:

- MPI Standard (available in PANDA)
- man-pages for required MPI methods
- Wikipedia on “Mandelbrot set”