Things to Try in MPI

Rather than giving detailed problems, this sheet suggests some things to try based on the example code provided. All of these things are possibly using nothing beyond the slides and notes. Some are trickier than others! Start where you need to and go as far as you think is useful

- Recap of Basic MPI

- First let's program a simple basic MPI code to recap
- Reading code is also good, so check out the examples in the Recap folders
- Skip this part if you don't need it

Try:

Write a code that runs on any number of processors, where each gets its rank and passes it to the rank one higher, looping around at the end so the highest rank sends to the lowest. Make sure to initialise and finalise your MPI correctly, and pay attention to avoiding deadlocks.

Check the performance of your program and make sure it scales OK. Note that you may need to run a lot of repeats of the pass to get a time you can observe.

- Domain Decomposition

- We mentioned the idea of minimising surface area (comms) to volume (calculation) - can we demonstrate this mattering in simple code?

Info:

Start with the code solving the heat equation, in the Domain_decomposition subfolder. Make sure this compiles and runs.

The `time` utility (https://linux.die.net/man/1/time) is a handy way of seeing how long a program takes to run

Try:

Run the code without Display. This can be done in the Fortran by compiling as shown in the README. In C, use `make noio`

Run the code on two different numbers of processors, using `time`. Adjust the number of points to take a measurable amount of time. Does the runtime drop as expected?

(Tricky) Try re-adjusting the code to only split the domain in 1 direction. How inefficient is this? Notice the “run to run” variation in timing - anything comparable to this isn’t going to be meaningful.

- Writing MPI code

- Adding MPI to a 1-D domain decomposed code

Try:

To get started with geometries, try parallelising the 1-D version of the heat equation code. Use the manual rank calculations or the MPI_Cart_* as you prefer.

How does this perform? You might find you need a very large domain to really get much scaling out of it

- Using MPI Types

- MPI types can both simplify comms and allow us to do things we otherwise never could
- They are

Info:

The examples folder Types contains examples of nearly all of the sorts of type MPI understands, with examples of creating them.

Try:

Have a look at these examples. Start with Struct and Contiguous

Try using a type in the 1-D domain decomposition. Does it help?

What sort of problems might really benefit from using a type?

- Other Examples

- Other models of MPI code

Info:

The Extras folder contains some examples of so-called “non-blocking” MPI

Try:

Have a look at these. How do they work?

Which problems do non-blocking comms like this fix? How does the non-blocking ring pass compare to using the combined sendrecv command?