|epcc|

Message-Passing Programming Tutorial Sheet

David Henty

1 Overlapping calculation and communication

We saw that one way to avoid deadlock when swapping halos was to use non-blocking communications, e.g. use a non-blocking send and then wait for it to complete later on. Between the send and the wait we simply issued a blocking receive, but in general it is possible also to perform calculations before waiting for the send operation to complete. In this way we can try and overlap communication and calculation, i.e. speed up the code by letting the send happen "in the background" while we do other useful work.

The aim is to identify calculations that can be performed without the need for communication. For the traffic modelling example, propose a way of overlapping the halo swaps with some of the calculations for the new road in terms of the old road. Can you design a communications library that makes it possible to switch between overlapped and non-overlapped communications without having to change the main code (i.e. only the library itself needs to be altered)? On what types of parallel machines do you think this overlapping approach might be most beneficial? Can you generalise your solution to the Case Study example?

2 Advanced use of collectives

Imagine you want to sum the values for each row and column in a matrix of size $M \times N$, i.e. in pseudocode:

- loop over $i = 1, M; j = 1, N$
  - $rowsum_i = rowsum_i + matrix_{i,j}$
  - $columnsum_j = columnsum_j + matrix_{i,j}$
- end loop

How would you parallelise this calculation so that rank 0 knows the row and column sums for the complete matrix in the following cases:

1. the matrix is decomposed across processes over one of its dimensions;
2. the matrix is decomposed across processes over both of its dimensions.

Consider whether you can do this using collective operations rather than point-to-point communications. For the second case you may want to create new communicators, e.g. look at the routine `MPI_Cart_sub`. Do not write any real code - just write pseudocode to explain the major steps in the calculation.

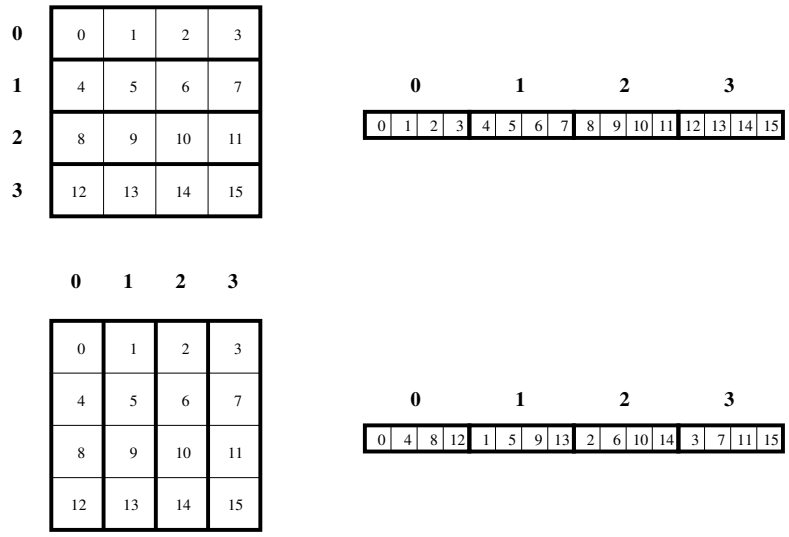

Figure 1: Assignment of data to processes for the two possible decompositions

3 Swapping between decompositions

We saw that the best way to divide up the Case Study problem was different in C and Fortran — we chose whether to split across the first or second dimension by considering which of these would ensure that the halo regions were contiguous sections of memory. Imagine now that we want to use both of these decompositions in the same program.

For example, in one part of the program we could be doing lots of sums over rows of the image, but performing sums over columns in another part. We therefore want to choose a decomposition that minimises communications for these sums by ensuring that entire rows or columns are stored on the same process. We will have to switch between decompositions within the code, which requires communications, but we hope that this will be more efficient overall than parallelising the sums over rows or columns.

You should first consider the case of a 4×4 image on 4 processes, where each process contains either a single row or column depending on the decomposition. If I number the 16 image pixels from 0 to 15, then in one decomposition process 0 will have elements 0,1,2 and 3, process 1 will have 4, 5, 6 and 7, etc. In the other decomposition process 0 will have 0, 4, 8 and 12, process 1 will have 1, 5, 9 and 13, etc. It may be helpful to look at Figure 1.

How could you use point-to-point communications to swap between these two decompositions, ensuring that you avoid deadlock? You should look at the collective routine `MPI_Alltoall` – how could this be used? Will you have to change your solutions for larger images where each process now has data from many rows or columns?

4 Decomposition

In the case study we only ever consider the situation where the domain can be split evenly, e.g. 8 processes can each have the same local size of 24 to make up a global size of 192. In general this will not be the case, and we would like to be able to run on, say, 10 processes.

I want you to decide how this should be done for a decomposition in strips (as used in the case study). For example, how would you divide up a problem size of 18 on 4 processes? Would you use (5,5,4,4), (4,4,5,5), (3,5,5,5), (5,5,5,3)? You should consider how the performance of the code is affected, but more importantly exactly how you would implement variable-sized domains in your image processing code. You need to consider not just how the halo swaps will work, but also how the use of collective communications in the IO phase (reading and writing the image) will be affected.

What do you think is the best choice of decomposition? What are your reasons for this choice?