Advanced Message-Passing Programming
Advanced derived datatypes practical

Alexei Borissov

29 January 2024

Introduction

This document contains the instructions for the practical related to the lecture "Advanced Derived Datatypes" in the module "Advanced Message-Passing Programming".

The objective of this practical is to get you to practice the creation of multiple MPI derived datatypes using the MPI datatype constructors covered in lectures.

Due to their brevity, all exercises can be run on the login nodes on Cirrus. Keep in mind however that good practice is to run jobs on compute nodes, using the job submission process.

You can do exercises in any order, although they have been designed to have an increasing level of difficulty.

Exercise 1. Scattering along non-contiguous dimensions

In this exercise, you are provided with a 2-dimensional array. In C, which is a programming language said to have a *row-major order*, elements consecutive in a row are also consecutive in memory, whereas column elements are not. Said otherwise, a row from that array is a contiguous chunk of memory, while a column is not.

Given the 2-dimension array declaration provided in Figure 1, and based on the above explanation about C memory order, it follows that elements `my_array[i][j]` and `my_array[i][j+1]` are consecutive in memory. There-

```c
int my_array[row_count][column_count];
```

Figure 1: Declaration of a 2-dimensional array in C.

fore, sending consecutive elements in the rightmost dimension is easy as a simple MPI contiguous datatype is sufficient.

In this exercise, the objective is to create an MPI derived datatype that represents the dimension that is not contiguous in memory and scatter it. In the code provided, MPI process 0 is the root and holds the entire array. All MPI processes will then define the MPI derived datatype representing a column using `MPI_Type_vector`. Finally, the root MPI process will scatter the 2-dimensional array across MPI processes by giving one column each.

Note As seen in lectures, remember that interleaving derived datatypes in MPI collectives involves the replication mechanism, itself requiring an extra step in order to properly calculate the displacements.

Exercise 2. Communicating structures

The second exercise consists of two MPI processes exchanging a variable that is of a compound type, which translates to a `struct` in C, which contains, in order: a `double`, an `integer` and a `character`.

The actual meaning of the underlying structure members is irrelevant in this practical. What matters is that they are specifically selected for particular educational purposes.

Case 1. In this first case, you are asked to construct an MPI derived datatype that can represent a variable of the structure type above and send it using a single `MPI_Send`. In other words, the objective is not to send each structure member in an individual send, doing so would mean running the risk to pay network latency as many times as there are structure members.

Case 2. The second case is almost identical to the first one. The only difference is simply a reordering of the members in the structure. Therefore, by copying the code you wrote to construct the derived datatype from case 1, it should be fairly straightforward to adapt it to this second case.

If your code is not working anymore, can you think of reasons that could explain why it is no longer working?

Exercise 3. Extensions

If you are seeking additional challenges, you can also consider the following extra exercises. They are entirely optional, and more difficult.

AoS to SoA. The array of structure (AoS) and structure of array (SoA) are two data structures whose usefulness and performance varies based on the context at hand. This extra exercise focusses on changing the layout of data such that you can move from an AoS to an SoA.

Try to construct two MPI derived datatypes such that you can send an array of elements having that structure. Then, receive it in such a way that it is stored in a structure of arrays where all first structure members are grouped together, where all second structure members are grouped together and so on.

Update the coursework. You are now able to use MPI derived datatypes in collective operations. As an additional challenge, rewrite the communications in your MPP coursework from last semester, and use vectors directly in scatters and gathers.