Set – MPI Topologies and Datatypes

Issued: December 14, 2022
Due Date: -

Question 1: Wave Equation

The wave equation is presented below

$$\frac{\partial^2 u}{\partial t^2} - c^2 \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} \right) = 0 \tag{1}$$

and it is often encountered in fluid dynamics or electromagnetics simulations. We are interested in solving eq. 1 for the following set of initial conditions

$$u(0, x, y, z) = f(x, y, z),$$
$$\frac{\partial u}{\partial t}(0, x, y, z) = 0, \tag{2}$$

where

$$f(x, y, z) = e^{-10r^2}$$

with $r = \sqrt{(x - 0.5)^2 + (y - 0.5)^2 + (z - 0.5)^2}$. We assume $c = 1$, a uniform mesh of $N_{tot} \times N_{tot} \times N_{tot}$ grid points and a cubic domain $[0, 1] \times [0, 1] \times [0, 1]$. Furthermore, we assume periodic boundary conditions, i.e.

$$u(t, 0, y, z) = u(t, 1, y, z),$$
$$u(t, x, 0, z) = u(t, x, 1, z), \tag{3}$$
$$u(t, x, y, 0) = u(t, x, y, 1).$$

In order to discretize eq.1 we apply a second order centered finite differences scheme in space and time. The discretized version of eq.1 is given below

$$u_{i,j,k}^{n+1} = 2u_{i,j,k}^n - u_{i,j,k}^{n-1} + c^2 \frac{\Delta t^2}{h^2}(u_{i+1,j,k}^n + u_{i-1,j,k}^n + u_{i,j+1,k}^n + u_{i,j-1,k}^n + u_{i,j,k+1}^n + u_{i,j,k-1}^n - 6u_{i,j}^n), \tag{4}$$

where $u_{i,j,k}^n = u(n\Delta t, (i + \frac{1}{2})h, (j + \frac{1}{2})h, (k + \frac{1}{2})h)$ is the function evaluation and $h = \frac{1}{N_{tot}}$ is the equidistant grid spacing.

You are given a parallel skeleton code (`wave.cpp`) that solves eq. 1 using eq. 4. The problem domain (unit cube) is decomposed into smaller, equal cubes of size $N \times N \times N$ (grid points). Each MPI rank operates on one of those cubes, which implies that the total number of ranks is s^3 , $s \in \mathbf{N}_+$. For an illustration see fig. 1, left.

Neighbouring ranks exchange cell data on the faces of their cubes at every time-step. More specifically, in order to apply eq. 4 next to a rank's boundary, it is necessary to receive a *face* of $N \times N$ points from a neighbouring rank, see fig. 1, right.
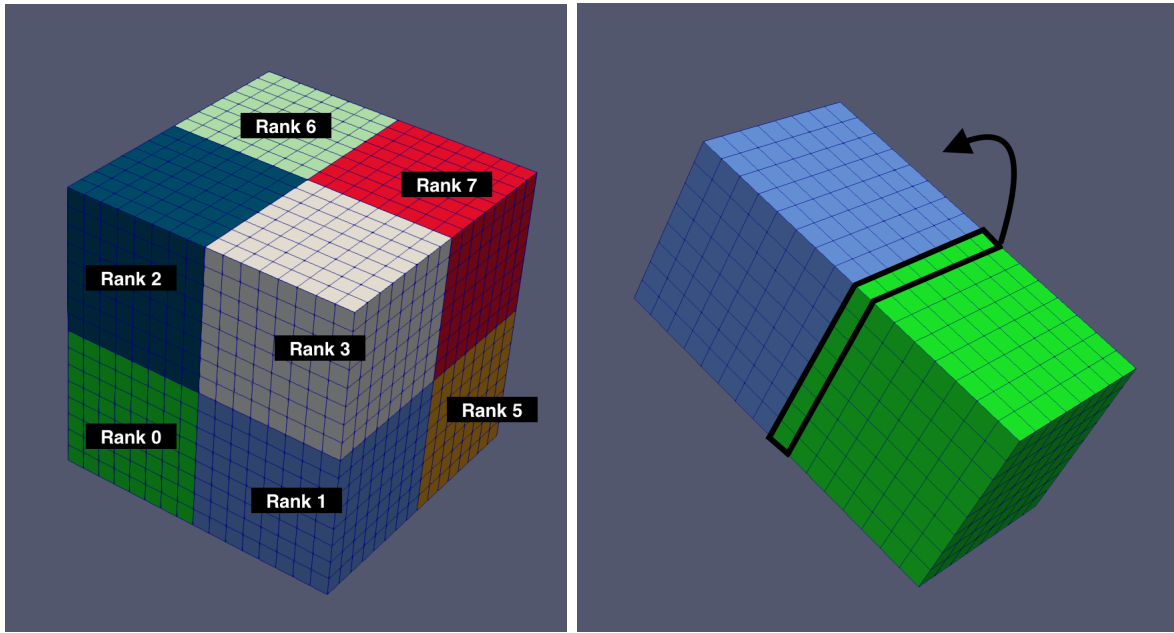
Figure 1: Domain decomposition of unit cube for 8 MPI ranks, $s = 2$ (left) and exchange of faces between neighbouring ranks (right). tabe ../xxx/xxx

a) A Cartesian MPI topology is suitable for this particular problem, as it can automatically compute the neighbours of every rank. Replace the complicated neighbour calculations in the skeleton code with a Cartesian MPI topology.

b) In order to exchange data among processes, a tedious process is followed in the skeleton code: Contiguous send buffers are first allocated, then data is manually collected and packed, before it is sent. Receive buffers are also allocated and after communication is complete data is manually unpacked and stored to the right location, before the computation is resumed.

All this can be avoided if custom MPI datatypes are used. Define custom MPI datatypes (hint: use MPI_Type_create_subarray) to send faces between processes and replace the previously described procedure.

Please consult the README file that is provided with the code for compilation instructions and further information.

Question 2: Cannon's algorithm

Cannon's algorithm[1] is a parallel algorithm for computing the product of two dense square matrices $C = A \times B$ in a series of $\sqrt{p}$ steps, where p is the number of ranks. Each rank owns a square sub-block of C and local sub-blocks of A and B. Each step rotates sub-blocks A_{ij} and B_{ij} along rows and columns of the 2D (i, j) processor geometry and computes a partial matrix product using the CBLAS *dgemm* operation to update its local portion $(C_{i,j} + = A_{i,j} \times B_{i,j})$, and then shift its submatrices in a ring-like fashion, as shown in Fig. 2.
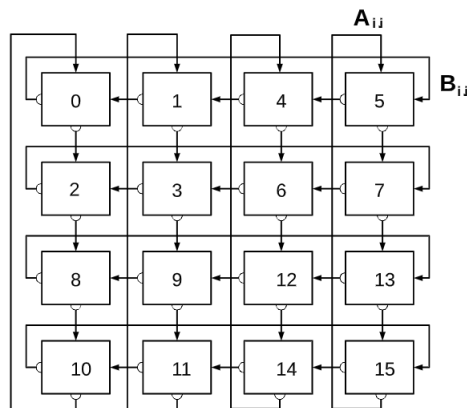

Figure 2: Ring-like communication topology used in Cannon's algorithm. Each square represents an MPI rank, and their position indicates their i,j submatrix coordinates. At each step, ranks compute $C_{i,j} + = A_{i,j} \times B_{i,j}$ and then shift their current $A_{i,j}$ submatrix downwards $B_{i,j}$ submatrix leftwards.

In this exercise, you are given a skeleton code that already implements Cannon's algorithm for matrix multiplication. However, the MPI topology is determined manually. This effort requires many lines of code and is bug-prone. Your task is to improve the code, using the advanced MPI tools you have seen in class.

a) Modify the code to use a Cartesian topology that automatically determines the neighbors of each MPI rank. (Hint: You may want to investigate whether specifying periodic grids is useful in this case).

b) Use a custom MPI Datatype to send and receive submatrices between ranks (Hint: Before you start, think about what type of MPI custom datatype would suffice in this case: *contiguous*, *vector*, or *struct*?).

---

[1] https://people.eecs.berkeley.edu/~demmel/cs267/lecture11/lecture11.html

Question 3: Send and receive custom structs

In this question you are asked to implement a custom MPI Datatype that allows two ranks to exchange different datatypes through a single message. One way to achieve this is to collect all datatypes to a struct, as seen below.

```
struct particle {
    int    id;
    double x[3];
    bool   state;
    double gamma;
};
```

You will only need to use two ranks; the goal is for rank 0 to successfully send the above struct to rank 1. Please refer to the provided skeleton code for more details.