High Performance Computing for Science and Engineering I

P. Koumoutsakos
ETH Zentrum, CLT E 13
CH-8092 Zürich

Fall semester 2019

Set 2 - MPI Topologies and Datatypes

Issued: March 09, 2020
Due Date: March 23, 2020 08:00am

The skeleton codes for this homework are located in the gitlab repository: https://gitlab.ethz.ch/hpcse20/exercise.

Question 1: Wave Equation (40 points)

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

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

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.

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

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

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

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

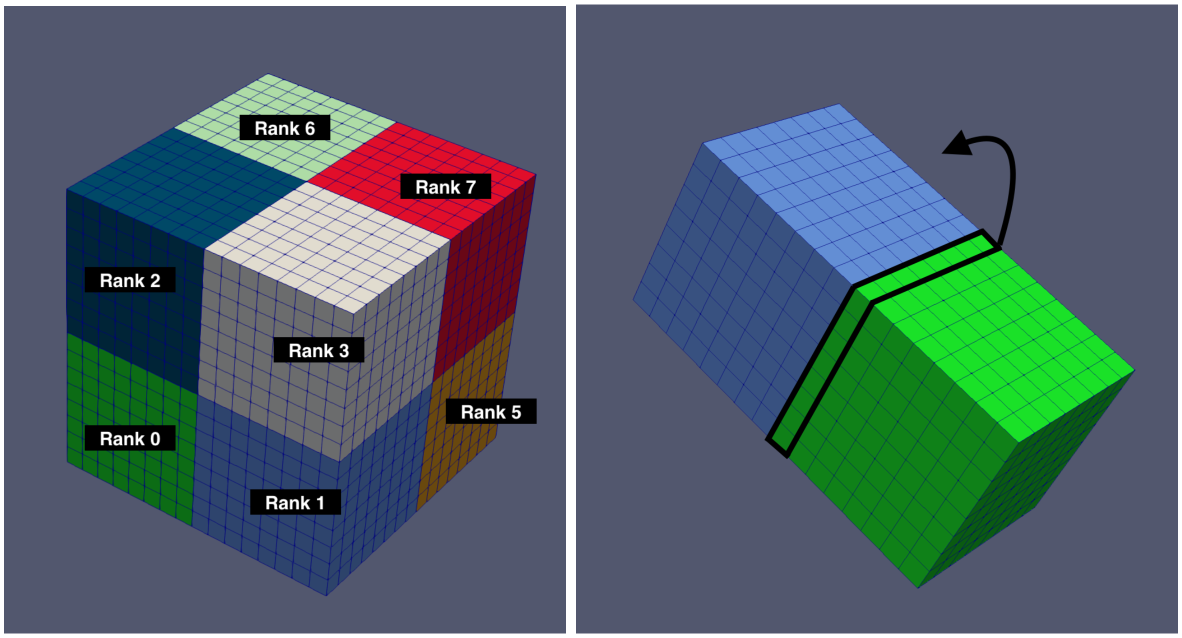

Figure 1: Domain decomposition of unit cube for 8 MPI ranks, $s = 2$ (left) and exchange of faces between neighbouring ranks (right).

You are given a parallel skeleton code 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.

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.

- 2 points for MPI_Cart_create
- 4 points for MPI_Cart_shift (called three times)
- 2 points for MPI_Cart_coords
- 2 points for MPI_Comm_free (called in struct destructor or after function run())

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.

- 1 point for each creation of each custom datatype (correct call of MPI_Type_create_subarray and MPI_Type_commit); (12 points total)

- 2 points for each correct pair of MPI_Isend and MPI_Irecv (correct message tags and rank IDs); (12 points total)
- 1 point for calling MPI_Type_free for each datatype, at the end of the simulation.

c) Now that your code is using a Cartesian topology, change your code in order to simulate a non-periodic problem with homogeneous Dirichlet boundary conditions

$$u(t, x, 0, z) = u(t, x, 1, z) = u(t, 0, y, z) = u(t, 1, y, z) = u(t, x, y, 0) = u(t, x, y, 1) = 0.$$

5 points if you can plot your solution and check that the boundary conditions are satisfied OR if you followed the approach introduced in the exercise session

- 3 points for changing the periodic array's entries to false, when creating the Cartesian communicator
- 2 points for changing the initial condition loop ranges, as shown in class

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

Question 2: Cannon's algorithm (10 points)

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.

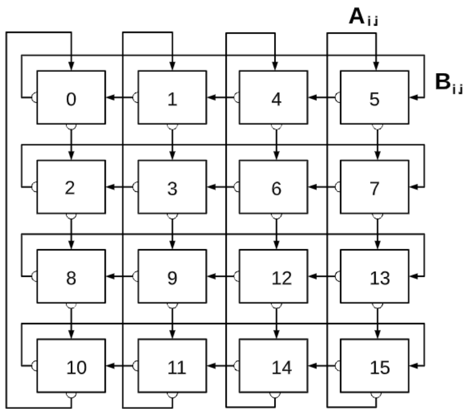

Figure 2: Ring-like communication topology used in Cannon's algorithm. Each square represents an MPI rank, and their position indicates their *i,j* submatrix coordinates. At each step, ranks compute $C_{i,j} + = A_{i,j} \times B_{i,j}$ and then shift their current $A_{i,j}$ submatrix downwards $B_{i,j}$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

- 2 points for MPI_Cart_create and MPI_Comm_free
- 1 point for MPI_Cart_shift
- 1 point for MPI_Cart_coords
- 2 points for using a periodic topology

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 point for using MPI_Type_contiguous
- 2 points for correct use of MPI_Isend and MPI_Irecv
- 1 point for MPI_Type_free

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

Question 3: Send and receive custom structs (10 points)

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.

- 6 points for using MPI_Type_create_struct or any other possibly non-portable approach that works and prints the correct results
- 4 points if you used MPI_Type_create_struct and MPI_Get_address to make your solution portable

Guidelines for reports submissions:

- Archive your source code (e.g.: .tar, .rar, .zip) and submit it via Moodle until March 23, 2020, 08:00am.