TDT4200 Parallel programming

PS3

Maren Wessel-Berg & Claudi Lleyda Moltó

September 2023

Practical information

Published: 19/09/23

Deadline: 03/10/23 at 22:00

Evaluation: Graded (10%)

- Completing the problem set is **mandatory** and it will count towards 10% of your final course grade.
- The work must be done **individually** and without help from anyone but the TDT4200 staff.
- **Reference** all sources found on the internet or elsewhere.
- The **requirements**, and **how and what to deliver** is explained in the problem set description found on BlackBoard.
- **Start early!**

Where can you get help with the assignment?

- **Recitation lecture**: introduction to the problem set

  (Today)
  Slides will be made available online.
- **TA hours**: ask questions in person

  Friday, September 22, 10:00–12:00 in Cybele
  Monday, September 25, 13:00–15:00 in Cybele
  Friday, September 29, 10:00–12:00 in Cybele
  Monday, October 02, 13:00–15:00 in Cybele
- **Piazza**: question forum

  Ask questions any time (but give us time to answer).
  Select the ps3 folder for questions related to this problem set.
  Do not post full or partial solutions!

Topic

Finite difference approximation of the 2D heat equation using MPI

- In PS2 you worked on a finite difference solver for the 2D heat equation and parallelised it using **MPI**.

  The goal was for you to get an introduction to MPI.
- In PS3 you will also work on a finite difference solver for the 2D heat equation, and you will also be using `MPI` for parallelisation.

  But we will use other MPI concepts and techniques.
- You will also **answer questions** about your implementation and the curriculum.

Today

- Introduce the problem set.
- (Talk about potential challenges that can arise when parallelising the FDM for the 2D heat equation.)

  These will be the same as last time, so we will not spend much time on it, but let me know if you want me to repeat something from last time.
- Repeat some MPI concepts from the main lectures.
- Introduce some MPI concepts not (yet?) introduced in the main lectures.

Your tasks

- Initialize and finalize the MPI environment
- Broadcast program arguments
- Decide on how to divide the grid into sub-grids and take care of memory allocation and domain initialization for each process
- Perform calculations in a sub-grid
- Communicate border values
- Handle program output with MPI I/O

Your tasks

- Initialize and finalize the MPI environment
- Broadcast program arguments
- Decide on how to divide the grid into sub-grids and take care of memory allocation and domain initialization for each process
- Perform calculations in a sub-grid
- Communicate border values
- Handle program output with MPI I/O

These are the same tasks as for PS2.

Your tasks

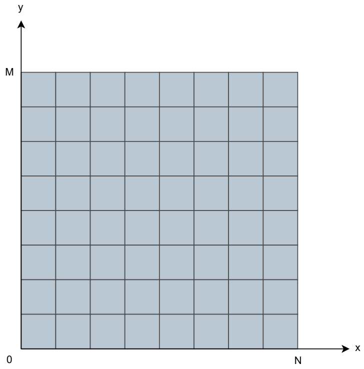
The difference this time is that we ask you to use a **cartesian communicator**.

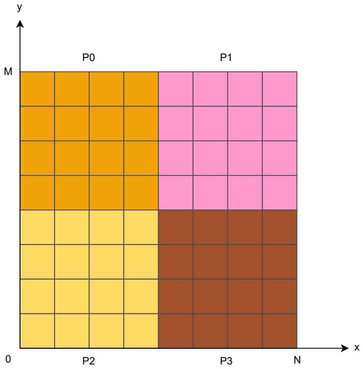
Your tasks

The difference this time is that we ask you to use a **cartesian communicator**.

This is convenient and might make certain things easier to implement, but there are some new concepts for you to be aware of.

Domain decomposition





Communicators

- A comminicator is a group of processes.

```
MPI_Comm comm;
```

- A rank is a unique number given to a process in a communicator to separate it from the others.

```
int rank;
MPI_Comm_rank ( comm, &rank );
```

- A process can be a member of more than one communicator, in which case its rank can be different in each communicator.
- A communicator will also have a size, which is the number of processes it contains.

```
int size;
MPI_Comm_size ( comm, &size );
```

Communicators

`MPI_COMM_WORLD`

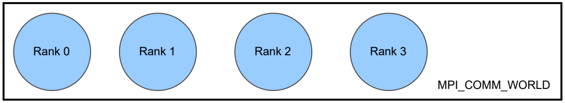
When you initialize the MPI environment with `MPI_Init` the `MPI_COMM_WORLD` communicator is set up, which contains all processes.

- The size of `MPI_COMM_WORLD` is the number you passed to mpirun with the `-np` option.

  ```
  mpirun -np p
  ```

- The rank of each process in `MPI_COMM_WORLD` is a number between 0 and p-1.

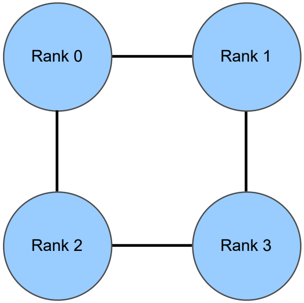
`mpirun -np 4` $\rightarrow$



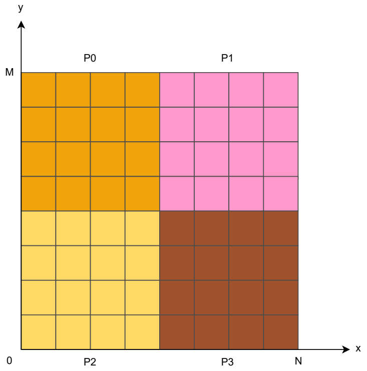
Communicators

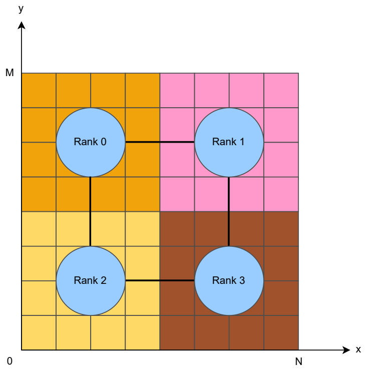
Cartesian

A *cartesian communicator* is a communicator where the **processes are organized in a grid**.



Domain decomposition and cartesian communicators





Creating a cartesian communicator

`MPI_Cart_create`

Creates a communicatior `commCart` from `commOld`.

```
MPI_Cart_create (
     MPI_Comm commOld,
     int nDims,
     int dims[],
     int periods[],
     int reorder,
     MPI_Comm *commCart
);
```

`nDims`: number of dimensions in the grid

`dims`: array of length `nDims` that specify the number processes in each dimension

`periods`: array of length `nDims` that specify if the dimension is periodic

`reorder`: can the processes get new ranks in `commCart` or do they need to be the same as in `commOld`?

Creating a cartesian communicator

`MPI_Dims_create`

How do we decide how many processeses there should be in each dimension?

```
MPI_Dims_create (
    int nProcs,
    int nDims,
    int dims[]
);
```

`nProcs`: the number of processes

`nDims`: the number of dimensions in the cartesian grid

`dims`: array of length `nDims` that contain the number of processes in each dimension

Accessing cartesian topology information

`MPI_Cart_coords`

```
MPI_Cart_coords (
    MPI_Comm commCart,
    int rank,
    int nDims,
    int coords[]
);
```

`commCart`: cartesian communicator

`rank`: rank of the process in `commCart`

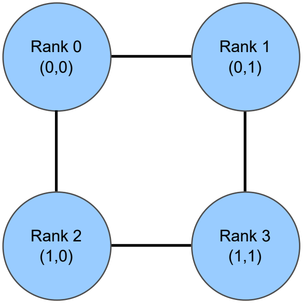
`nDims`: number of dimensions in the cartesian grid

`coords`: array of size `nDims` containing the cartesian coordinates of the process with rank `rank`

Accessing cartesian topology information

`MPI_Cart_coords`

```
MPI_Cart_coords (
    MPI_Comm commCart,
    int rank,
    int nDims,
    int coords[]
);
```



Accessing cartesian topology information

`MPI_Cart_shift`

```
MPI_Cart_shift (
    MPI_Comm commCart,
    int direction,
    int displacement,
    int *rankSource,
    int *rankDestination
);
```

`commCart`: cartesian communicator

`direction`: in which dimension are we moving?

`displacement`: how much are we moving?

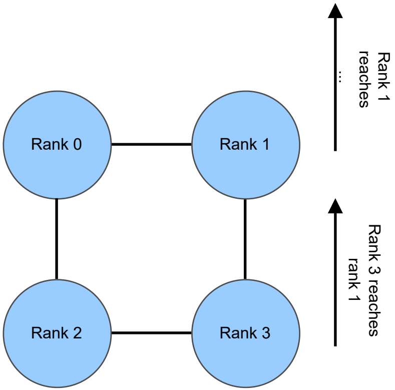
`rankSource`: which rank will reach the calling process?

`rankDestination`: which rank will the calling process reach?

Accessing cartesian topology information

`MPI_Cart_shift`

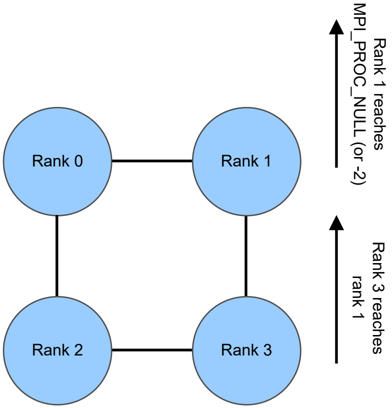
Example 1:
Rank 1 calls `MPI_Cart_Shift` with `direction`=0 and `displacement`=1. `rankSource` will be `MPI_PROC_NULL` and `rankDestination` will be 3 (as in rank 3).



Accessing cartesian topology information

`MPI_Cart_shift`

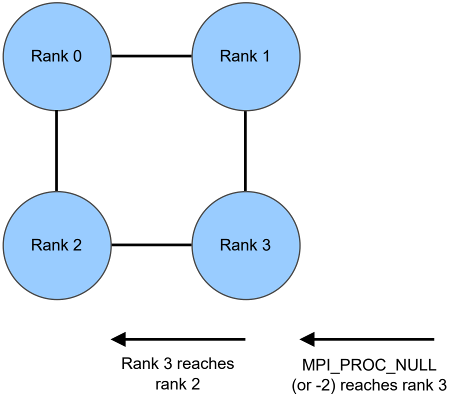
Example 1:
Rank 1 calls `MPI_Cart_Shift` with `direction`=0 and `displacement`=1. `rankSource` will be `MPI_PROC_NULL` and `rankDestination` will be 3 (as in rank 3).



Accessing cartesian topology information

`MPI_Cart_shift`

Example 2:
Rank 3 calls `MPI_Cart_Shift` with `direction`=1 and `displacement`=1. `rankSource` will be 2 (as in rank 2) and `rankDestination` will be `MPI_PROC_NULL`.



Derived datatypes

- In the problem set description we have indicated for which tasks it might be helpful to **create your own datatypes**.
- It is not a requirement, but **strongly recommended.**
- The main lectures have covered MPI datatypes and derived datatypes, so I will not repeat it today (`https://folk.idi.ntnu.no/janchris/tdt4200-au23/slides/slides14.pdf`).

Evaluation

MPI implementation: 70%

Code clairity and documentation: 10%

Questions: 20%

- The check assumes that the domain borders are not written to file.
- The check is meant as a helpful tool, but running with different number of processes, plotting, and inspecting the plots can also give useful information about any bugs.
- You do not need to document code that you have not written.
- Partial solutions will also get points :)

Trouble accessing home directory when connected to Snotra

- Some of you could not access your home directory when connected to the Snotra cluster.
- The problem should be fixed now.
- If you still cannot access your home directory, run `kinit && cd` after connecting.
- This is also stated at the top of the terminal window after you connect.

(You can do this exercise on your own pc.)

Extra resources

Documentation (and nice examples): RookieHPC

Debugging: tmpi