MAX-PLANCK-INSTITUT
FÜR PLASMAPHYSIK

Multi Regions-of-Interest
in REFMUL3

Tiago Ribeiro[1]
Filipe da Silva[2]
Jorge Santos[2]
[1]MPI für Plasmaphysik [2]IPFN/IST U. Lisboa

EUROfusion

This work has been carried out within the framework of the EUROfusion Consortium, funded by the European Union via the Euratom Research and Training Programme (Grant Agreement No 101052200 — EUROfusion). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the European Commission can be held responsible for them.

TÉCNICO LISBOA

Model and numerics

Solve for Maxwell's curl equations + constitutive relation (current density)

$$\nabla \times \mathbf{H} = \varepsilon_0 \frac{\partial \mathbf{E}}{\partial t} + \sigma \mathbf{E} + \mathbf{J}$$

$$\nabla \times \mathbf{E} = -\mu_0 \frac{\partial \mathbf{H}}{\partial t} - \sigma^\star \mathbf{H}$$

$$\frac{\partial \mathbf{J}}{\partial t} = \varepsilon_0 \omega_p^2 \mathbf{E} + \vec{\omega}_c \times \mathbf{J}$$

using the Finite Difference Time Domain (FDTD) method in Cartesian meshes

`REFMUL3` code

3D full-wave C code: hybrid OpenMP/MPI (3D) + OpenMP target offload (GPU)

K.S. Yee, *IEEE Trans Antennas Propag* **14** (1966) 302
L. Xu and N. Yuan *IEEE Antennas Wirel Propag Lett* **5** (2006) 335

F. Silva, S. Heuraux and T. Ribeiro *Proc. IRW13* (2017)
F. Silva, S. Heuraux, T. Ribeiro et al. *Fus Eng Des* **202** (2024) 114354

Motivation

What are regions-of-interest (ROI)?

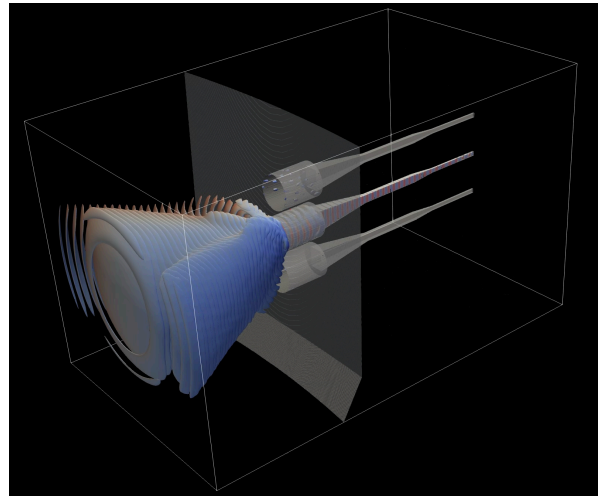
- wherever waves propagate!

Why do we need to consider them?

- wave-free volumes are currently computed
  - outside the vessel
  - inside metalic structures

Avoid:

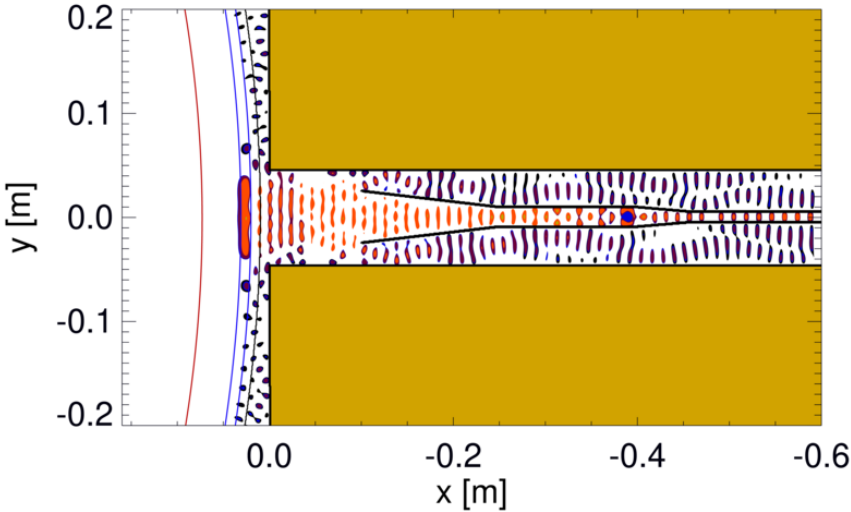
Waste of resources!



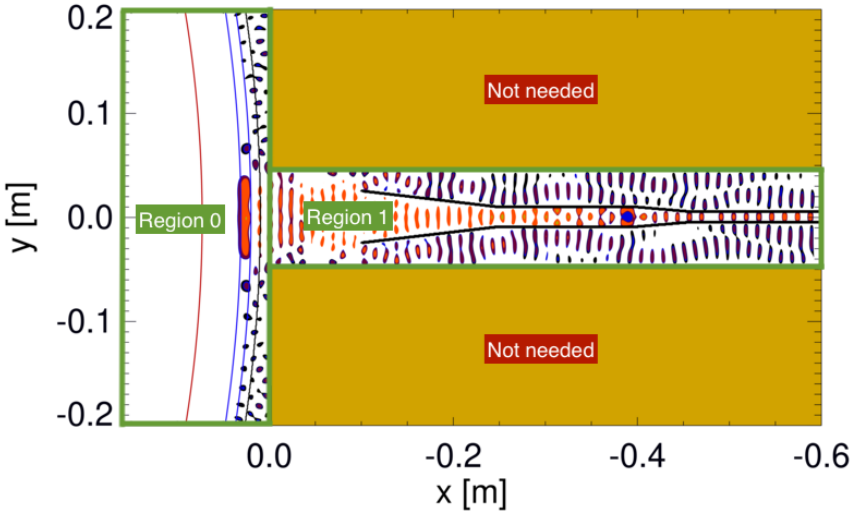
Outline

Outline

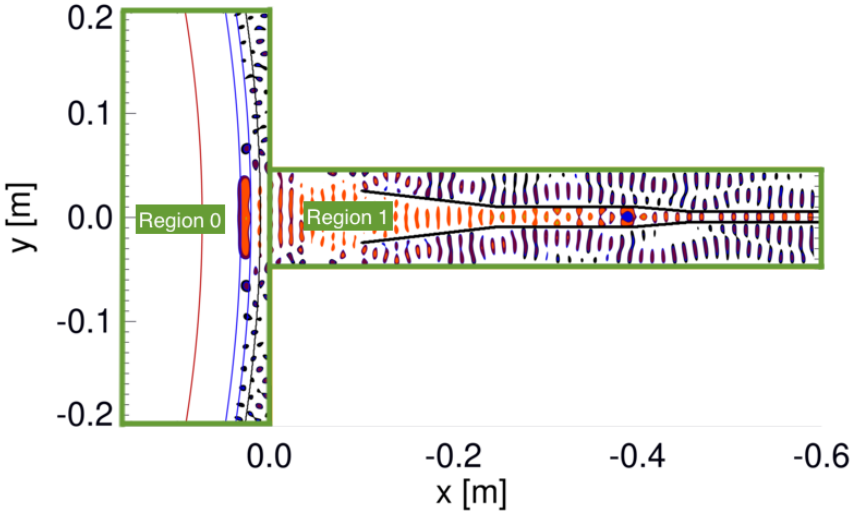
Example in 2D (simpler to illustrate)



Example in 2D (simpler to illustrate)



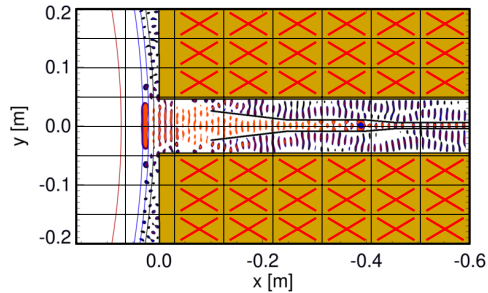
Example in 2D (simpler to illustrate)



How to proceed?

A) Remove void subdomains

- Preprocess the decomposition
- Find/remove empty subdomains
- Caution: MPI Cartesian topology
- Issue: low granularities (e.g. GPUs)



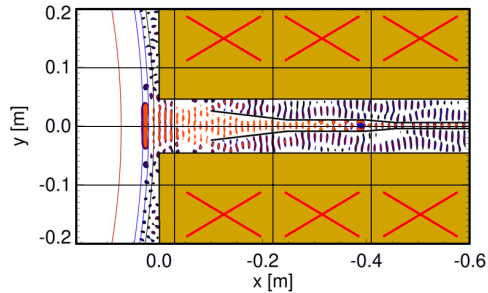
B) Define regions-of-interest (ROI)

- Global virtual space
- More effective
- Same grid-resolution for all ROIs
- Challenge: non-matching subdomains

How to proceed?

A) Remove void subdomains

- Preprocess the decomposition
- Find/remove empty subdomains
- Caution: MPI Cartesian topology
- Issue: low granularities (e.g. GPUs)



B) Define regions-of-interest (ROI)

- Global virtual space
- More effective
- Same grid-resolution for all ROIs
- Challenge: non-matching subdomains

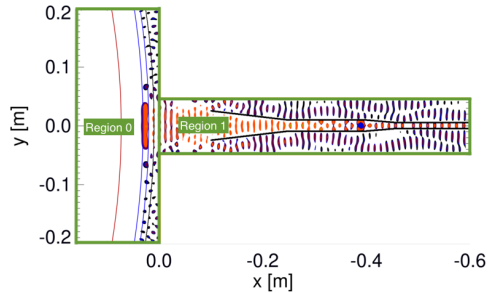
How to proceed?

A) Remove void subdomains

- Preprocess the decomposition
- Find/remove empty subdomains
- Caution: MPI Cartesian topology
- Issue: low granularities (e.g. GPUs)

B) Define regions-of-interest (ROI)

- Global virtual space
- More effective
- Same grid-resolution for all ROIs
- Challenge: non-matching subdomains



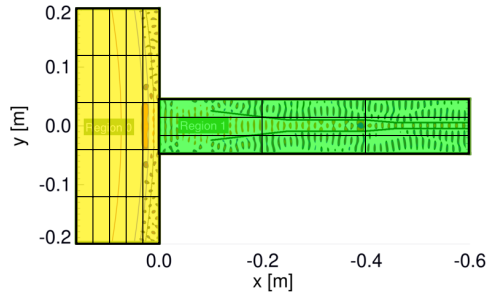
How to proceed?

A) Remove void subdomains

- Preprocess the decomposition
- Find/remove empty subdomains
- Caution: MPI Cartesian topology
- Issue: low granularities (e.g. GPUs)

B) Define regions-of-interest (ROI)

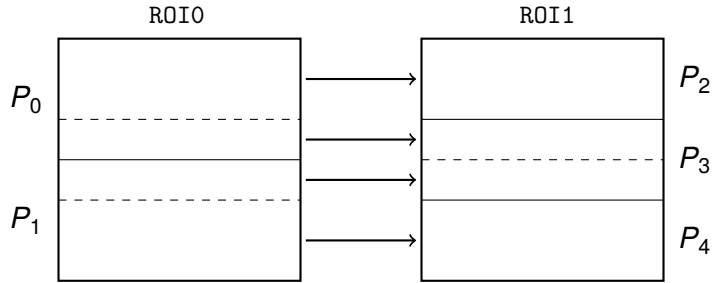
- Global virtual space
- More effective
- Same grid-resolution for all ROIs
- Challenge: non-matching subdomains



Outline

Starting point: non-matching subdomains

2D data sent from left to right, 1D domain decomposition (inspired in [1])



Beyond simple point-to-point communication

Each MPI task sends different messages to multiple MPI tasks

[1] *Using Advanced MPI: Modern Features of the Message-Passing* (2014) by W. Gropp, T. Hoefler, R. Thakur and E. Lusk

MPI collective communication function: `MPI_Alltoallv()`

```
1 int MPI_Alltoallv(
2   void *sendbuf, int *sendcnts, int *sendispls, MPI_Datatype sendtype,
3   void *recvbuf, int *recvcnts, int *recvdispls, MPI_Datatype recvtype,
4   MPI_Comm comm);
```

Parameters

- `sendbuf`: pointer to the send buffer containing the data to be sent.
- `sendcnts`: integer array (of length group size) specifying the number of elements to send to each task.
- `sendispls`: integer array (of length group size) specifying the displacement of first element relative to `sendbuf`.
- `sendtype`: data type of send buffer elements (handle).
- `recvbuf`: address of receive buffer (choice).
- `recvcnts`: integer array (of length group size) specifying the number of elements to receive from each task.
- `recvdispls`: integer array (of length group size) specifying the displacement of first relative to `recvbuf`.
- `recvtype`: data type of receive buffer elements (handle).
- `comm`: communicator (handle).

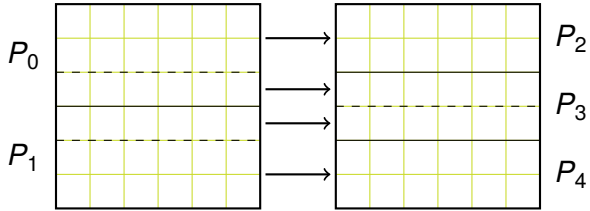
MPI collective communication function: `MPI_Alltoallv()`

```
1 int MPI_Alltoallv(
2   void *sendbuf, int *sendcnts, int *sendispls, MPI_Datatype sendtype,
3   void *recvbuf, int *recvcnts, int *recvdispls, MPI_Datatype recvtype,
4   MPI_Comm comm);
```

Parameters

- `sendbuf`: pointer to the send buffer containing the data to be sent.
- `sendcnts`: integer array (of length group size) specifying the number of elements to send to each task.
- `sendispls`: integer array (of length group size) specifying the displacement of first element relative to `sendbuf`.
- `sendtype`: data type of send buffer elements (handle).
- `recvbuf`: address of receive buffer (choice).
- `recvcnts`: integer array (of length group size) specifying the number of elements to receive from each task.
- `recvdispls`: integer array (of length group size) specifying the displacement of first relative to `recvbuf`.
- `recvtype`: data type of receive buffer elements (handle).
- `comm`: communicator (handle).

Communication send patterns: 6x6 grid



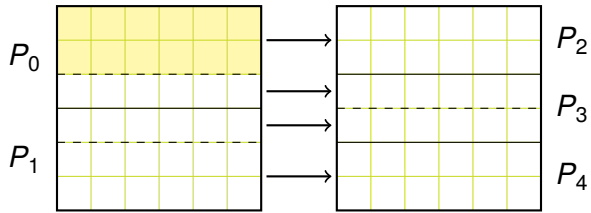
`sendcnts[ntasks]`

task	0	1	2	3	4
0					
1					
2					
3					
4					

`sendispls[ntasks]`

task	0	1	2	3	4
0					
1					
2					
3					
4					

Communication send patterns: 6x6 grid



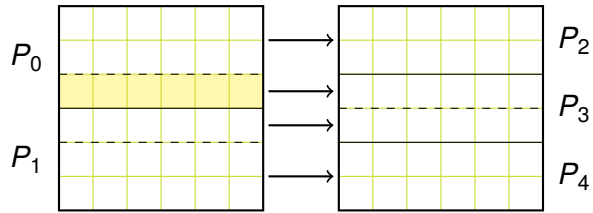
`sendcnts[ntasks]`

task	0	1	2	3	4
0	0	0	12		
1					
2					
3					
4					

`sendispls[ntasks]`

task	0	1	2	3	4
0	0	0	0		
1					
2					
3					
4					

Communication send patterns: 6x6 grid



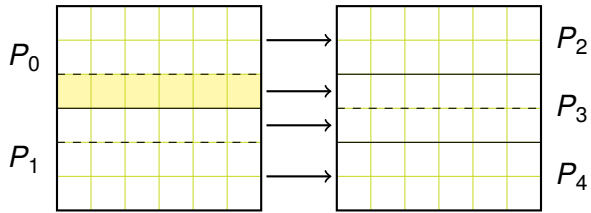
`sendcnts[ntasks]`

task	0	1	2	3	4
0	0	0	12	6	
1					
2					
3					
4					

`sendispls[ntasks]`

task	0	1	2	3	4
0	0	0	0	12	
1					
2					
3					
4					

Communication send patterns: 6x6 grid



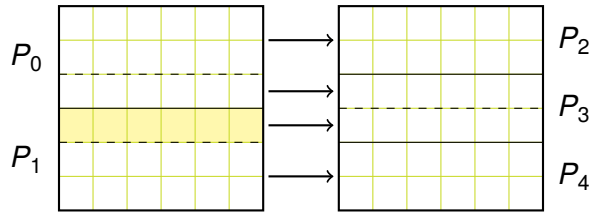
`sendcnts[ntasks]`

task	0	1	2	3	4
0	0	0	12	6	0
1					
2					
3					
4					

`sendispls[ntasks]`

task	0	1	2	3	4
0	0	0	0	12	0
1					
2					
3					
4					

Communication send patterns: 6x6 grid



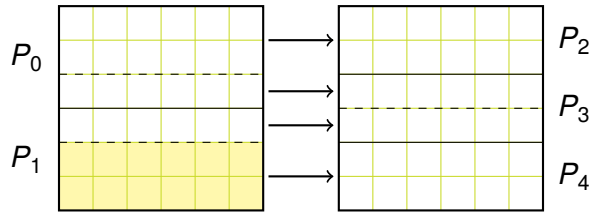
`sendcnts[ntasks]`

task	0	1	2	3	4
0	0	0	12	6	0
1	0	0	0	6	
2					
3					
4					

`sendispls[ntasks]`

task	0	1	2	3	4
0	0	0	0	12	0
1	0	0	0	0	
2					
3					
4					

Communication send patterns: 6x6 grid



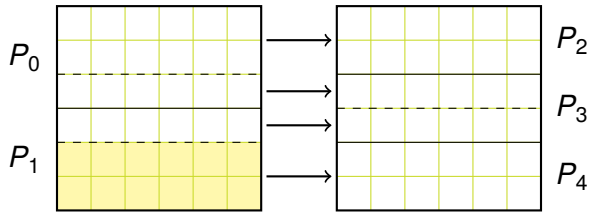
`sendcnts[ntasks]`

task	0	1	2	3	4
0	0	0	12	6	0
1	0	0	0	6	12
2					
3					
4					

`sendispls[ntasks]`

task	0	1	2	3	4
0	0	0	0	12	0
1	0	0	0	0	6
2					
3					
4					

Communication send patterns: 6x6 grid



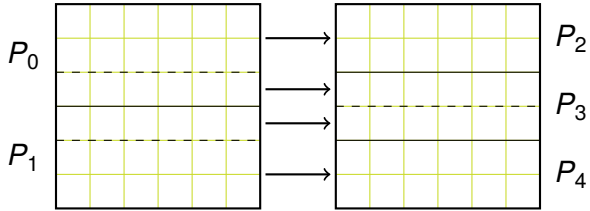
`sendcnts[ntasks]`

task	0	1	2	3	4
0	0	0	12	6	0
1	0	0	0	6	12
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0

`sendispls[ntasks]`

task	0	1	2	3	4
0	0	0	0	12	0
1	0	0	0	0	6
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0

Communication receive patterns: 6x6 grid



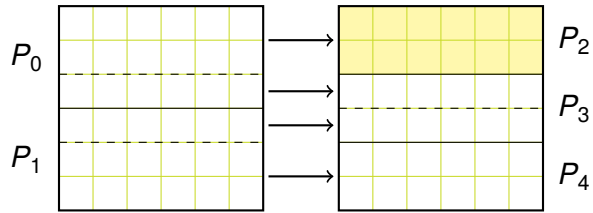
`recvcnts[ntasks]`

task	0	1	2	3	4
0					
1					
2					
3					
4					

`recvispls[ntasks]`

task	0	1	2	3	4
0					
1					
2					
3					
4					

Communication receive patterns: 6x6 grid



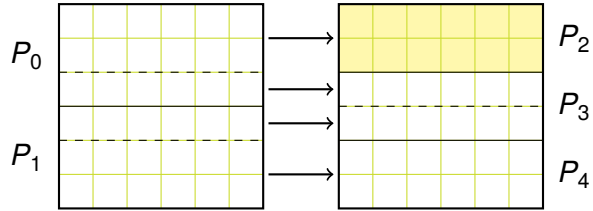
`recvcnts[ntasks]`

task	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	12				
3					
4					

`recvispls[ntasks]`

task	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	0				
3					
4					

Communication receive patterns: 6x6 grid



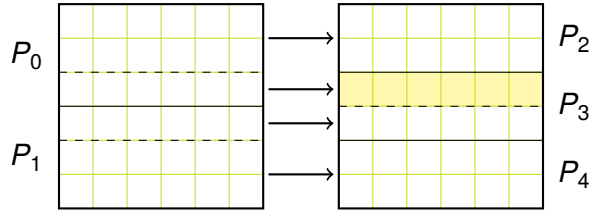
`recvcnts[ntasks]`

task	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	12	0	0	0	0
3					
4					

`recvispls[ntasks]`

task	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3					
4					

Communication receive patterns: 6x6 grid



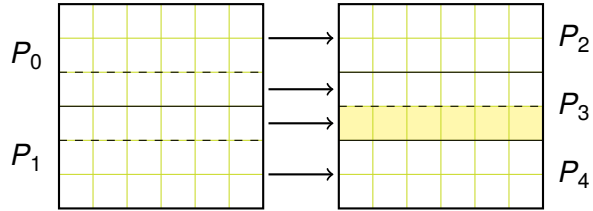
`recvcnts[ntasks]`

task	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	12	0	0	0	0
3	6				
4					

`recvispls[ntasks]`

task	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0				
4					

Communication receive patterns: 6x6 grid



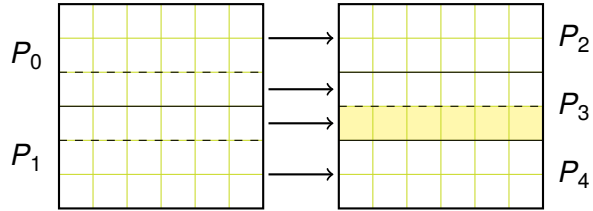
`recvcnts[ntasks]`

task	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	12	0	0	0	0
3	6	6			
4					

`recvispls[ntasks]`

task	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	6			
4					

Communication receive patterns: 6x6 grid



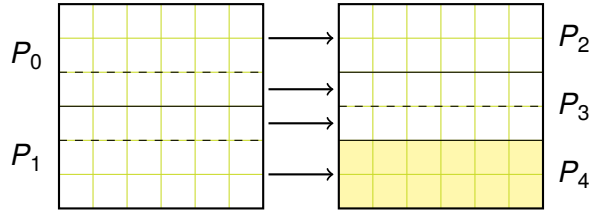
`recvcnts[ntasks]`

task	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	12	0	0	0	0
3	6	6	0	0	0
4					

`recvispls[ntasks]`

task	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	6	0	0	0
4					

Communication receive patterns: 6x6 grid



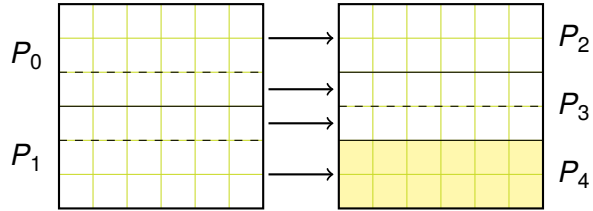
`recvcnts[ntasks]`

task	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	12	0	0	0	0
3	6	6	0	0	0
4	0	12			

`recvispls[ntasks]`

task	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	6	0	0	0
4	0	0			

Communication receive patterns: 6x6 grid



`recvcnts[ntasks]`

task	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	12	0	0	0	0
3	6	6	0	0	0
4	0	12	0	0	0

`recvispls[ntasks]`

task	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	6	0	0	0
4	0	0	0	0	0

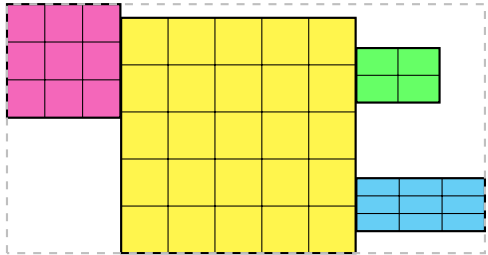
Compute arguments for `MPI_Alltoallv()`

```
1  /* loop ranks in remote ROI only, since there are no local ROI exchanges */
2  for (int iRank = myRankF[color_rm]; iRank < myRankL[color_rm]; iRank++) {
3    if (color == color_here) {
4      jF_rm = calc_jF(iRank - myRankF[color_rm], nRanks_ROI[color_rm], Ni, Nj);
5      jL_rm = calc_jL(iRank - myRankF[color_rm], nRanks_ROI[color_rm], Ni, Nj);
6      if (jF <= jL_rm && jL >= jF_rm) {
7        commcounts[iRank] = ( min(jL, jL_rm) - max(jF, jF_rm) + 1) * Ni;
8        commispls[iRank] = displs;
9        displs += commcounts[iRank];}
10     else {
11       commcounts[iRank] = 0;
12       commdispls[iRank] = 0;}
13   } else {  /* color == color_rm */
14     commcounts[iRank] = 0;
15     commdispls[iRank] = 0;}
16 }
```

Computing `*counts` & `*displs` is easier than using point-to-point communication

Outline

Generalizations in 2D

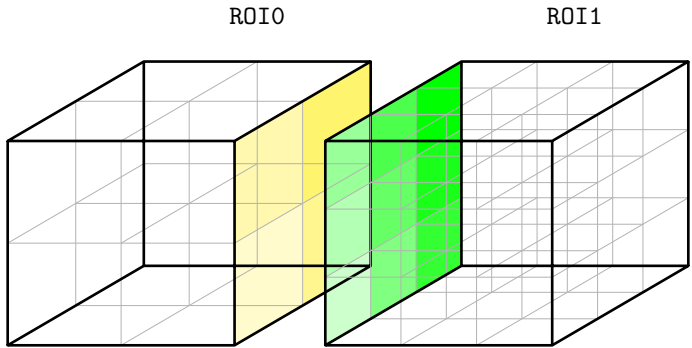


- 2D domain decomposition: adjust computation of `counts` & `displs`
- Exchange of ghost-layers: communication buffers become 1D
- MPI communicators & groups: subset of MPI tasks in each ROI interface
- $>$ 2 ROIs: global virtual space (not allocated) as reference frame

General 2D domain decomposed multi-ROI configurations: implemented and tested

Outline

Towards 3D

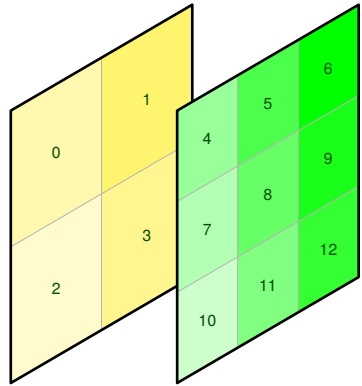


Important difference:

Communication buffers between 3D ROIs are 2D faces decomposed in 2D

Interface communication

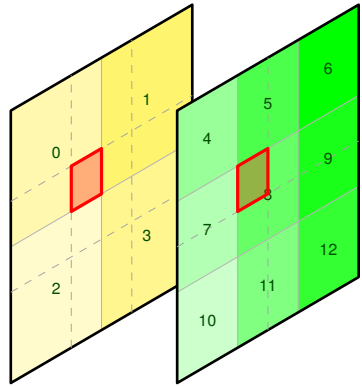
2D communication buffers



Interface communication

2D communication buffers

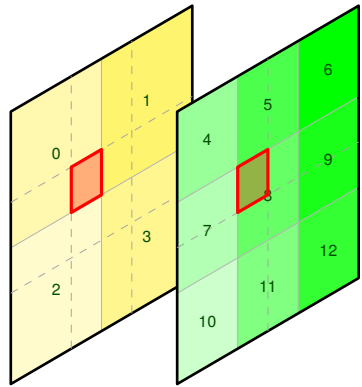
- Exchanged data: subsets of 2D data
- Memory access pattern is non-contiguous!



Interface communication

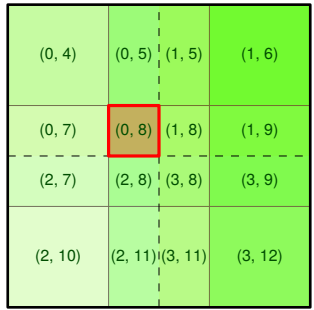
2D communication buffers

- Exchanged data: subsets of 2D data
- Memory access pattern is non-contiguous!
- Arrays of `counts` and `displs` + `MPI_Alltoallv()`: not enough
- Further concepts required, e.g. `blocksize` and `stride`



Non-contiguous memory access

How can we generalize our algorithm?



superposition of interface communication buffers and data layout

Solution:

Use most general `MPI_Alltoallw()` call together with MPI derived data types

MPI derived data types + `MPI_Alltoallw()` (ongoing)

```
1 int MPI_Alltoallw(
2   void *sendbuf, int *sendcnts, int *sendispls, MPI_Datatype *sendtype,
3   void *recvbuf, int *recvcnts, int *recvdispls, MPI_Datatype *recvtype,
4   MPI_Comm comm);
```

Strategy

- Compute `counts` and `displs` as before (using Cartesian topology)
- Use this information to create array of MPI derived data types (DDT)
- Each DDT describes memory access pattern to communication buffers
- Call `MPI_Alltoallw()` (not `v`!) using array of DDTs as argument

Note:

Already implemented for 2D multi-ROI case, extension to 3D ongoing...

Outline

Summary & outlook

Multi regions-of-interest

- Same grid-resolution for all ROIs
- Non-matching interface subdomains
- Communication algorithm devised
- Tailored all-to-all communication + memory access patterns
- General multi-ROI 2D configuration: implemented
- Extension to 3D ongoing: should be finished soon...

Outlook

- Deployment to `REFMUL3`: left for future work