MAX-PLANCK-INSTITUT
FÜR PLASMAPHYSIK

Multi Regions-of-Interest in REFMUL3

non-matching subdomain communication

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

Outline

REFMUL3 code
- Model and numerics
- Motivation

Multi regions-of-interest (ROI)

Communication patterns I
- Simplest case in 2D

Communication patterns II
- Non-contiguous memory access
- Derived datatypes
- Interface communicators

Prototype code multi_roi-3d

Summary & outlook

Outline

REFMUL3 code
- Model and numerics
- Motivation

Multi regions-of-interest (ROI)

Communication patterns I
- Simplest case in 2D

Communication patterns II
- Non-contiguous memory access
- Derived datatypes
- Interface communicators

Prototype code `multi_roi-3d`

Summary & outlook

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)?

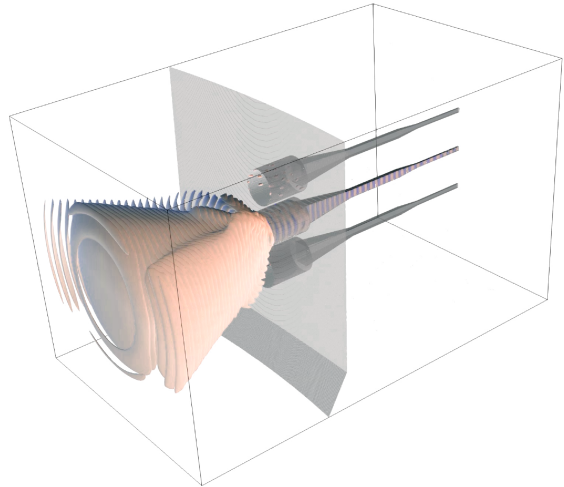
- wherever waves propagate!

Why do we need to consider them?

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

Avoid:

Waste of resources!



Outline

REFMUL3 code
- Model and numerics
- Motivation

Multi regions-of-interest (ROI)

Communication patterns I
- Simplest case in 2D

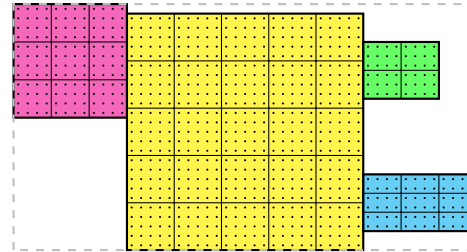
Communication patterns II
- Non-contiguous memory access
- Derived datatypes
- Interface communicators

Prototype code `multi_roi-3d`

Summary & outlook

Multi regions-of-interest (ROI): the concept

- Global virtual domain: not allocated
- ROIs around computational regions
- Same grid-resolution for all ROIs
- Independent ROI decompositions
- Generalization to 3D



Challenges

- Subdomain shapes: exchange of different messages with multiple neighbors
- Multiple interfaces per ROI: complex hierarchy of communicators

Outline

REFMUL3 code
- Model and numerics
- Motivation

Multi regions-of-interest (ROI)

Communication patterns I
- Simplest case in 2D

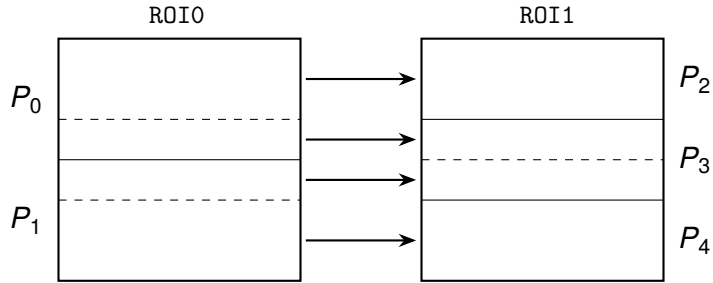
Communication patterns II
- Non-contiguous memory access
- Derived datatypes
- Interface communicators

Prototype code `multi_roi-3d`

Summary & outlook

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()`

```
int MPI_Alltoallv(
  void *sendbuf, int *sendcnts, int *sendispls, MPI_Datatype sendtype,
  void *recvbuf, int *recvcnts, int *recvdispls, MPI_Datatype recvtype,
  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`: datatype 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`: datatype of receive buffer elements (handle).
- `comm`: communicator (handle).

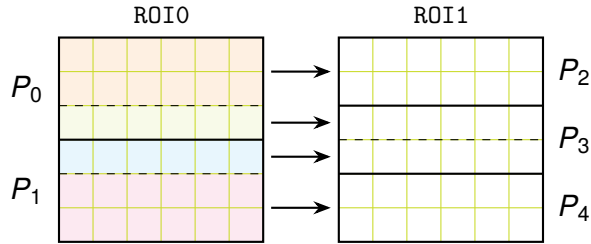
MPI collective communication function: `MPI_Alltoallv()`

```
int MPI_Alltoallv(
  void *sendbuf, int *sendcnts, int *sendispls, MPI_Datatype sendtype,
  void *recvbuf, int *recvcnts, int *recvdispls, MPI_Datatype recvtype,
  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`: datatype 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`: datatype of receive buffer elements (handle).
- `comm`: communicator (handle).

Communication patterns: ROI0 sends to ROI1 (6x6 grid)



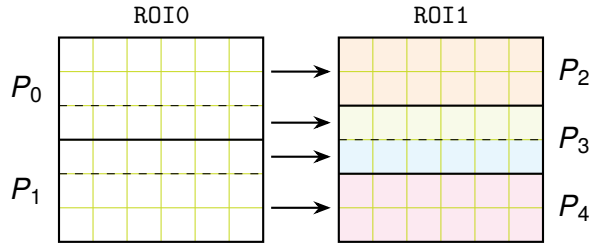
`sendcnts[ntasks]`

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 patterns: ROI0 receives from ROI1 (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

`recvdispls[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

Communication symmetry

Full data exchange between ROIs in 2 steps: swap `counts/displs` arguments

`ROI0` sends to `ROI1` & `ROI1` receives from `ROI0`

- `MPI_Alltoallv(..., sendcounts/senddispls, recvcnts/recvdispls,...)`

`ROI1` sends to `ROI0` & `ROI0` receives from `ROI1`

- `MPI_Alltoallv(..., recvcounts/recvdispls, sendcounts/senddispls,...)`

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

Outline

REFMUL3 code
- Model and numerics
- Motivation

Multi regions-of-interest (ROI)

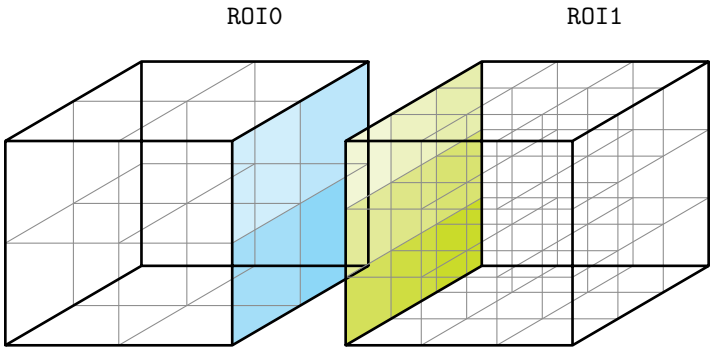
Communication patterns I
- Simplest case in 2D

Communication patterns II
- Non-contiguous memory access
- Derived datatypes
- Interface communicators

Prototype code `multi_roi-3d`

Summary & outlook

2D interfaces in 3D domains

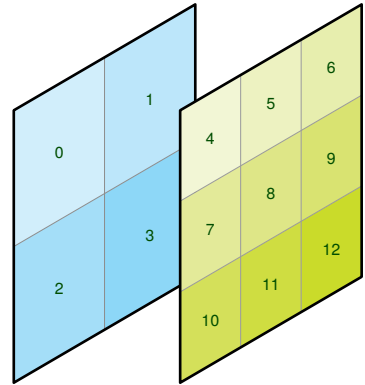


Important difference:

Communication buffers are 2D interfaces decomposed independently in 2D

Interface communication

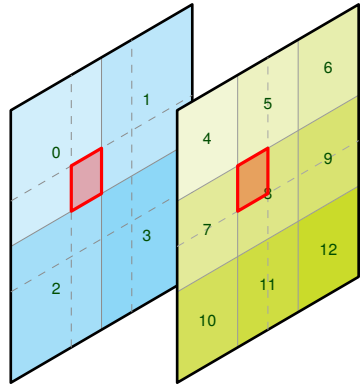
2D buffers, 2D decomposition



Interface communication

2D buffers, 2D decomposition

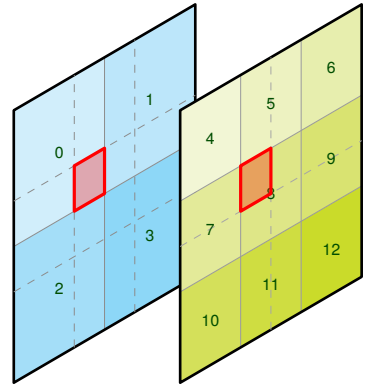
- Messages: subsets of 2D data
- Memory access pattern is non-contiguous!
- Arrays of `counts` and `displs` + `MPI_Alltoallv()`: not enough
- Further concepts required, e.g. `blocksize` and `stride`



Interface communication

2D buffers, 2D decomposition

- Messages: 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`



Solution:

Use MPI derived datatypes together with more general `MPI_Alltoallw()`

MPI collective communication function: `MPI_Alltoallw()`



```
int MPI_Alltoallw(
  void *sendbuf, int *sendcnts, int *sendispls, MPI_Datatype *sendtypes,
  void *recvbuf, int *recvcnts, int *recvdispls, MPI_Datatype *recvtypes,
  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`.
- `sendtypes`: datatype array (of length group size) specifying the type of data to send to each task (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`.
- `recvtypes`: datatype array (of length group size) specifying the type of data to receive from each task (handle).
- `comm`: communicator (handle).

Outline

REFMUL3 code
- Model and numerics
- Motivation

Multi regions-of-interest (ROI)

Communication patterns I
- Simplest case in 2D

Communication patterns II
- Non-contiguous memory access
- Derived datatypes
- Interface communicators

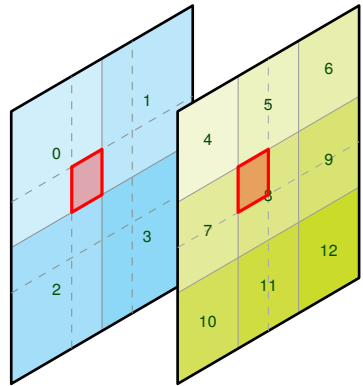
Prototype code multi_roi-3d

Summary & outlook

MPI derived datatypes

Non-contiguous memory access patterns to communication buffers

- Requires use of derived datatypes
- Implementation steps:
  - "what shape?"
    use `counts` in each direction
    to define new `type_aux` (1st level)
  - "starting where?"
    use `displs` in each direction + `type_aux`
    to define final `send/recvtype` (2nd level)



MPI derived datatypes: pseudo-code

For each rank in the interface ([nRanks_interface])

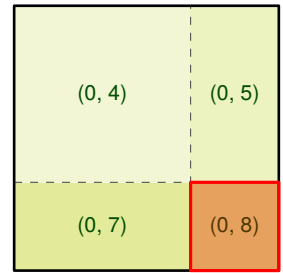
- Compute (send/recv) counts/displs (as before) separately for fast/slow indexes (1D)
- 1st level

```
MPI_Type_vector(count = counts_slow
                blocklen = counts_fast
                stride = Nfast_par
                otype = MPI_double
                newtype = send/recvtype_aux)
```

- 2nd level

```
MPI_Type_create_hindexed(count = 1
  blocklen = min(counts_fast * counts_slow, 1)
  displ = displs_slow * Nfast_par + displs_fast
  otype = send/recvtype_aux
  newtype = send/recvtype)
```

Overlap between ranks 0 and 4,5,7,8



Communication symmetry II

Actual communication step:

Use new derived datatypes array as argument for `MPI_Alltoallw()`

```
MPI_Alltoallw(*send/recvbuf = communication buffers (same as before),
              *send/recvcnts = 1s or 0s (for exchange or not)
              *send/recvdispls = 0s (contained in DDT)
              *send/recvtypes = new derived datatypes (shape+location),
              comm_interface = communicator restricted to interface
```

Call twice with swapped `send`/`recv` arguments

For exchange in both directions (e.g., ROI0 $\leftrightarrow$ ROI1)

Outline

REFMUL3 code
- Model and numerics
- Motivation

Multi regions-of-interest (ROI)

Communication patterns I
- Simplest case in 2D

Communication patterns II
- Non-contiguous memory access
- Derived datatypes
- Interface communicators

Prototype code multi_roi-3d

Summary & outlook

Interface communicators: concept

- Sub-communicator for each ROI: `MPI_Comm_split()`
- 3D Cartesian topology via `MPI_Cart_create()`: define ROI faces
- Detect ROI interfaces – who touches whom
  ROIs share spatial coordinates with everyone
- Convert ROI interface ranks to world ranks
  broadcast to everyone because ROI communicators are local
- Join matching ROI interface ranks into interface group:
  `MPI_Group_incl()` & `MPI_Group_union()`
- Convert to `comm_interface` per interface: `MPI_Comm_create_group`
  ranks outside that interface get `MPI_COMM_NULL`

Better performance

Restrict all-to-all complexity

Outline

REFMUL3 code
- Model and numerics
- Motivation

Multi regions-of-interest (ROI)

Communication patterns I
- Simplest case in 2D

Communication patterns II
- Non-contiguous memory access
- Derived datatypes
- Interface communicators

Prototype code `multi_roi-3d`

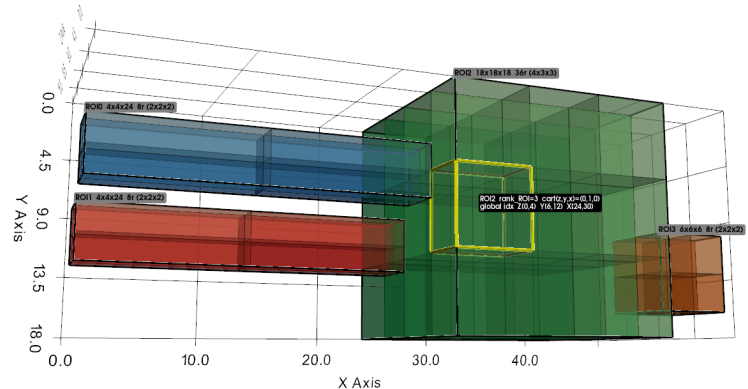
Summary & outlook

Prototype code `multi_roi-3d`

- Written in `C` using `CMake` build system
- Extends the (single-ROI) domain decomposition logic of `REFMUL3`
- Adds MPI communication between arbitrary number of ROIs
- Replaces the numerical kernel with single 3D static field generator
- Built-in end-to-end tests verify communication against ground truth
- Unit tests via `CMocka`
- Documentation via `Doxygen` and `Sphinx`
- CI pipeline via `GitLab CI/CD`
  - builds/runs code (including end-to-end tests)
  - runs unit tests
  - generates documentation

PyVista helper tool (Claude)

```
python roi_setup.py --roi  4 4  24   8  4 4 0  --roi 4 4 24  8  9 9  0 \
                    --roi 18 18 18  36  0 0 24 --roi 6 6 6   8  1 10 42
```



Outline

REFMUL3 code
- Model and numerics
- Motivation

Multi regions-of-interest (ROI)

Communication patterns I
- Simplest case in 2D

Communication patterns II
- Non-contiguous memory access
- Derived datatypes
- Interface communicators

Prototype code `multi_roi-3d`

Summary & outlook

Summary & outlook

Multi regions-of-interest approach: prototype code `multi_roi-3d`

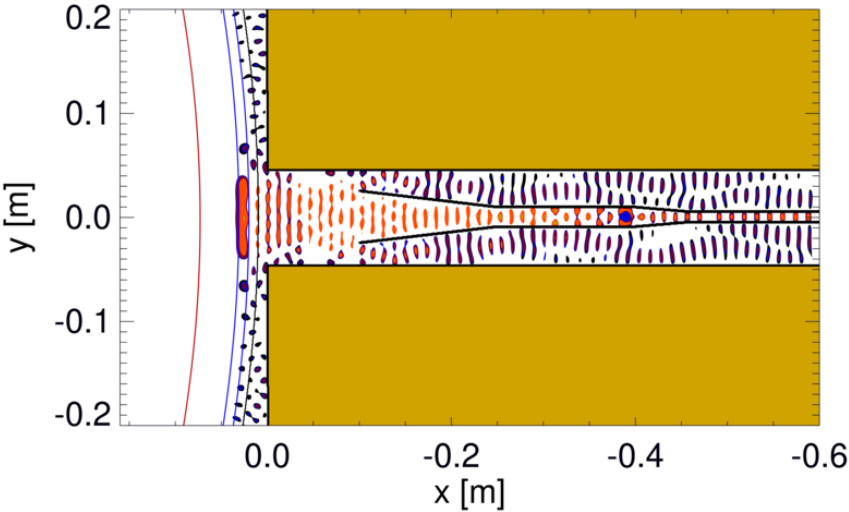
- Virtual global domain (not allocated) contains all ROIs (arbitrary number)
- ROIs enclose regions where waves propagate (allocated)
- Same grid-resolution for all ROIs, independent domain decomposition for each
- MPI communicators hierarchy: intra-ROI (`REFMUL3`) + inter-ROI (new)
  - Non-matching inter-ROI subdomain interfaces
  - Tailored collective communication: `MPI_Alltoallw` + derived datatypes
  - Communicator hierarchy –> interface communicators

Outlook

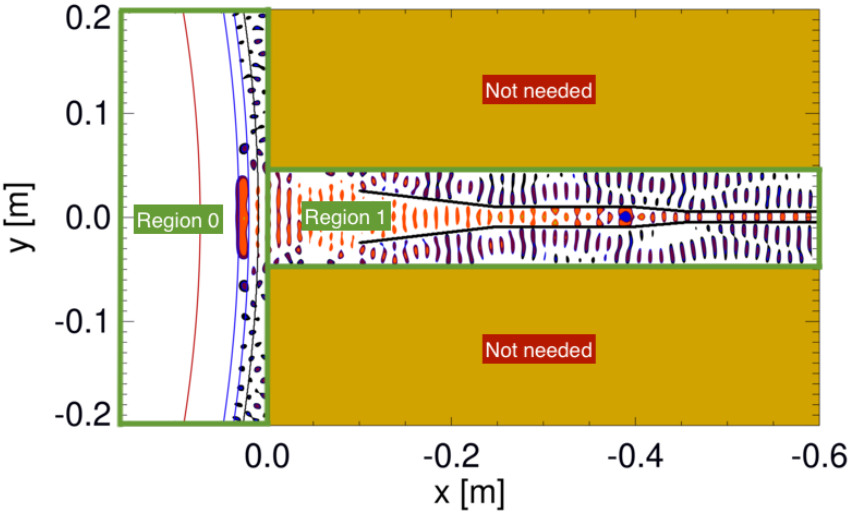
- Deployment to `REFMUL3`: next step
- Different ROI grid resolutions: longer term possibility

Backup slides

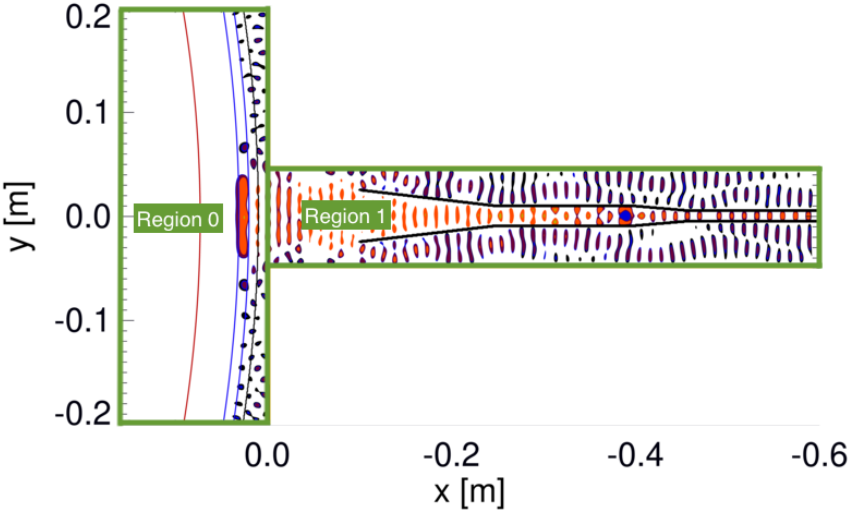
Our approach: simplified 2D example



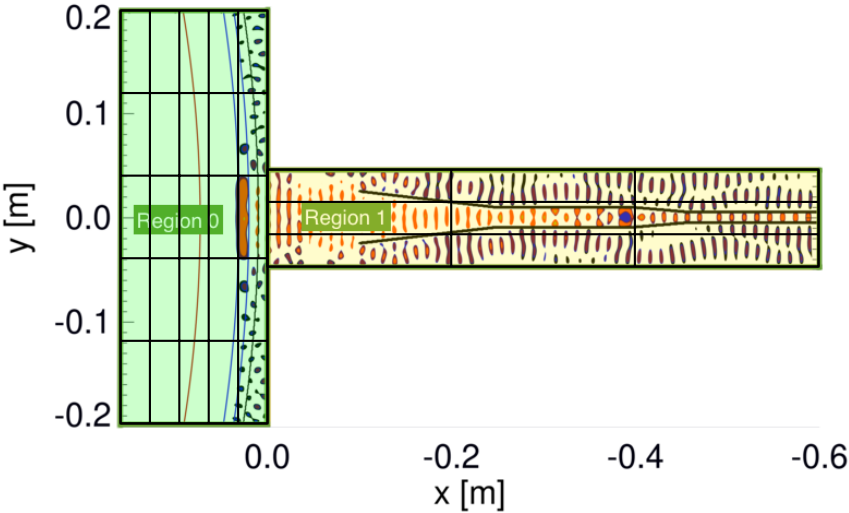
Our approach: simplified 2D example



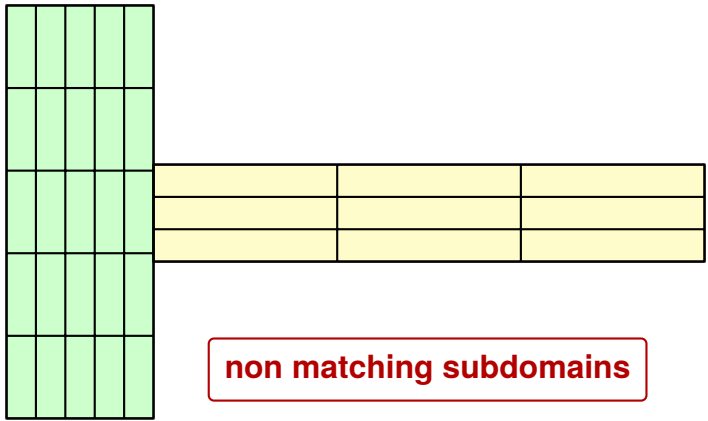
Our approach: simplified 2D example



Our approach: simplified 2D example



Our approach: simplified 2D example



Compute arguments for `MPI_Alltoallv()`

```
/* loop ranks in remote ROI only, since there are no local ROI exchanges */
for (int iRank = myRankF[color_rm]; iRank < myRankL[color_rm]; iRank++) {
  if (color == color_here) {
    jF_rm = calc_jF(iRank - myRankF[color_rm], nRanks_ROI[color_rm], Ni, Nj);
    jL_rm = calc_jL(iRank - myRankF[color_rm], nRanks_ROI[color_rm], Ni, Nj);
    if (jF <= jL_rm && jL >= jF_rm) {
      commcounts[iRank] = ( min(jL, jL_rm) - max(jF, jF_rm) + 1) * Ni;
      commispls[iRank] = displs;
      displs += commcounts[iRank];}
    else {
      commcounts[iRank] = 0;
      commdispls[iRank] = 0;}
  } else {  /* color == color_rm */
    commcounts[iRank] = 0;
    commdispls[iRank] = 0;}
}
```

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

Communication patterns

