DEUTSCHES KLIMARECHENZENTRUM

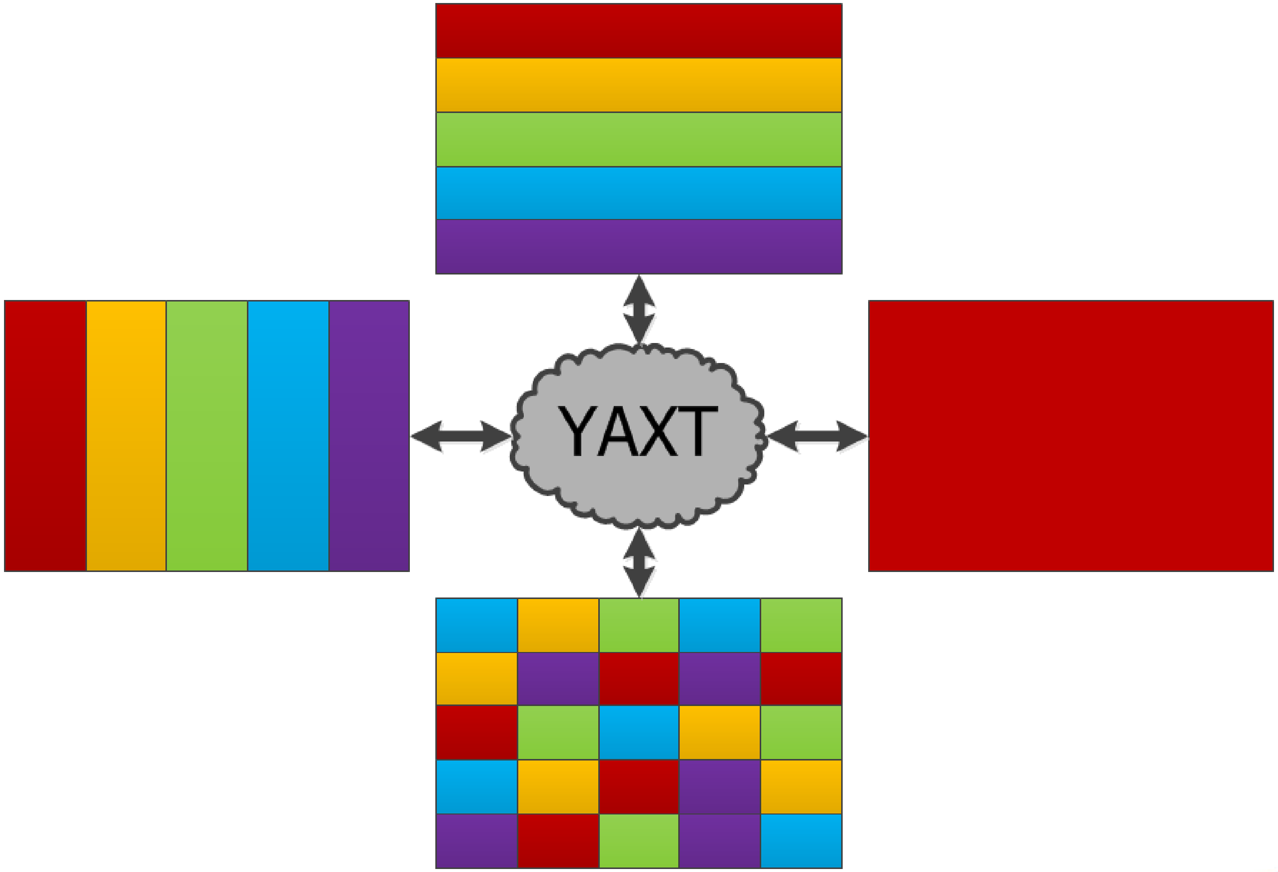
Introduction to YAXT

(Yet Another eXchange Tool, DKRZ 2012-2013)

Jörg Behrens, Moritz Hanke, Thomas Jahns

What it does

- Redistribution of data between two sets of processes



Jörg Behrens, Moritz Hanke,
Thomas Jahns

Some remarks

- Library on top of MPI
- Inspired by Fortran Prototype *Unitrans* by Mathias Pütz in ScalES-project
- Implemented in C ⇒ type invariant
- Fully-featured Fortran interface (requires C-interop)
- Supported by DKRZ
- BSD license
- Git and SVN-readonly mirror repositories available
- Uses pkg-config (`pkg-config --cflags yaxt`)

How it works

1. Initialisation
   - Each process defines what data it has and what it wants (source and target decomposition).
   - Generate a mapping between source and target decomposition.
   - Generate data type specific redistribution object for a given mapping.
2. Timeloop
   - Do the exchange
3. Finalisation
   - Clean up

Jörg Behrens, Moritz Hanke, Thomas Jahns

Defining a decomposition

- Assumptions:
  - All data elements have the same memory layout and are stored in an array (C makes no difference between 1D and nD-arrays)
  - Each data object has a unique global id (integer).
- A decomposition is a list of global data element ids.
- The positions of the global ids within the set correspond to the positions of the respective data elements within the data array (if nothing else is said).

Generate mapping between src and tgt decomposition

- To generate a mapping you need to provide the two decompositions and a MPI communicator.
- The operation is collective for all processes within the given communicator.

Generate data specific redistribution object

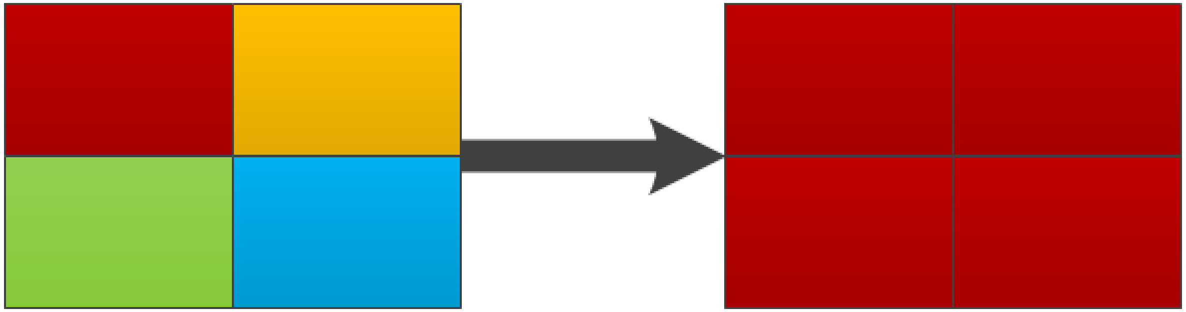
- To generate a data specific redistribution object the user needs to provide a mapping object and the MPI data type for the data elements.
- It is possible to combine multiple existing redistribution objects (useful for aggregation of data exchanges).

Doing the exchange and clean up

- To do the exchange the user needs to call the exchange routine and provide a redistribution object and the source and target array(s) (passed via C_LOC from Fortran).
- All objects generated by YAXT can be destroyed by calling the respective destructor.

Example 1

Gather on rank 0



Defining a decomposition.

```
int src_index = rank;
Xt_idxlist src_idxlist =
  xt_idxvec_new(&src_index, 1);
Xt_idxlist tgt_idxlist;
if (rank == 0) {
  struct Xt_stripe tgt_stripe =
  {.start = 0, .nstrides = 4, .stride = 1};
  tgt_idxlist =
      xt_idxstripes_new(&tgt_stripe, 1);
} else
  tgt_idxlist = xt_idxempty_new();
```


Jörg Behrens, Moritz Hanke, Thomas Jahns

Generate mapping between src and tgt decomposition.

```
Xt_xmap xmap =
   xt_xmap_all2all_new(
      src_idxlist, tgt_idxlist,
      MPI_COMM_WORLD);
```

Generate data specific redistribution object.

```
Xt_redist redist =
  xt_redist_p2p_new(xmap, MPI_INT);
```


Jörg Behrens, Moritz Hanke, Thomas Jahns

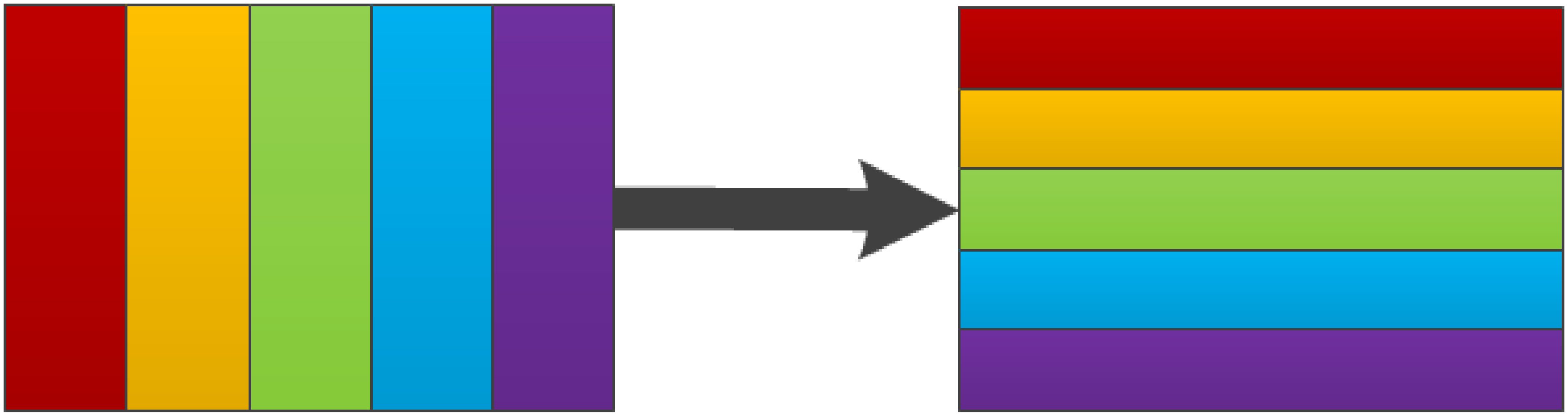
Doing the exchange and clean up.

```
int src_data[1] = {…};
int tgt_data[4];

xt_redist_s_exchange1(
  redist, src_data, tgt_data);

xt_redist_delete(redist);
xt_xmap_delete(xmap);
xt_idxlist_delete(tgt_idxlist);
xt_idxlist_delete(src_idxlist);
```


Jörg Behrens, Moritz Hanke, Thomas Jahns

Example 2

Transpose from row-wise to column-wise decomposition



Example program: `examples/row2col_f.f90`

Create idxlist for row-wise decomposition

```
row_part_range(1, 1) = &
     INT(uniform_partition_start(global_range(:, 1), &
     comm_size, rank + 1))
row_part_range(2, 1) = &
     INT(uniform_partition_start(global_range(:, 1), &
     comm_size, rank + 2)) - 1
row_part_range(:, 2) = INT(global_range(:, 2))
row_part_shape(:) = row_part_range(2, :) - &
     row_part_range(1, :) + 1
```

Create idxlist for column-wise decomposition

```
col_part_range(:, 1) = INT(global_range(:, 1))
col_part_range(1, 2) = &
    INT(uniform_partition_start(global_range(:, 2), &
    comm_size, rank + 1))
col_part_range(2, 2) = &
    INT(uniform_partition_start(global_range(:, 2), &
    comm_size, rank + 2)) - 1
col_part_shape(:) = col_part_range(2, :) &
    - col_part_range(1, :) + 1
```

example for 10x5 global shape and 2nd (= MPI rank 1) of 3 processes

```
global_range(1:2, 1:2) = RESHAPE((/ 1, 10, 1, 5, &
                                 (/ 2, 2 /))
row_part_range(1, 1) = 4
row_part_range(2, 1) = 7
row_part_range(:, 2) = (/ 1, 5 /)
row_part_shape(:) = (/ 4, 5 /)
col_part_range(:, 1) = (/ 1, 10 /)
col_part_range(1, 2) = 2
col_part_range(2, 2) = 3
col_part_shape(:) = (/ 10, 2 /)
```


Jörg Behrens, Moritz Hanke, Thomas Jahns

Create decomposition descriptors

```
src_idxlist = xt_idxfsection_new(0_xt_int_kind, &
    INT(global_shape, xt_int_kind),             &
    INT(row_part_shape, xt_int_kind),           &
    INT(row_part_range(1, :), xt_int_kind))
tgt_idxlist = xt_idxfsection_new(0_xt_int_kind, &
    INT(global_shape, xt_int_kind),             &
    INT(col_part_shape, xt_int_kind),           &
    INT(col_part_range(1, :), xt_int_kind))
```


Jörg Behrens, Moritz Hanke,
Thomas Jahns

Create mapping and redistribution

```
! generate exchange map
xmap = xt_xmap_all2all_new(src_idxlist, tgt_idxlist, &
    mpi_comm_world)

! generate redistribution object
redist = xt_redist_p2p_new(xmap, mpi_double_precision)
```


Jörg Behrens, Moritz Hanke,
Thomas Jahns

Fill source array and redistribute

```
! prepare arrays
ALLOCATE(src_array(row_part_range(1, 1):row_part_range(2, 1),  &
     &             row_part_range(1, 2):row_part_range(2, 2)), &
     &   tgt_array(col_part_range(1, 1):col_part_range(2, 1),  &
     &             col_part_range(1, 2):col_part_range(2, 2)))

DO j = row_part_range(1, 2), row_part_range(2, 2)
  DO i = row_part_range(1, 1), row_part_range(2, 1)
    src_array(i, j) = DBLE(i * j)
  END DO
END DO

! do the exchange
CALL xt_redist_s_exchange1(redist, C_LOC(src_array), C_LOC(tgt_array))
```


Jörg Behrens, Moritz Hanke, Thomas Jahns

Finish up

```
! clean up
 DEALLOCATE(tgt_array, src_array)
 CALL xt_redist_delete(redist)
 CALL xt_xmap_delete(xmap)
 CALL xt_idxlist_delete(tgt_idxlist)
 CALL xt_idxlist_delete(src_idxlist)

 ! finalise
 CALL xt_finalize()
 CALL mpi_finalize(ierror)
 IF (ierror /= mpi_success) STOP 1
```

Methods to define a decomposition

- Index lists (Xt_idxlist) describe subsets of indices within the global index space
- The global index space can be any list of indices
- There are multiple methods for constructing index lists:
  - Index vector
  - Index stripes
  - Index section
  - Index list collection
  - Index modifier
  - Empty Index list

Methods to define a decomposition

- Index vector
  - Arbitrary list of indices:
    - [0,3,32,26,44,14,48]
- Index stripes
  - List of stripes:
    [[start = 0, nstrides = 3, stride = 2], [start = 1, nstrides = 2, stride = 2]] == [0,2,4,1,3]
- Index section
  - N-dimensional block of indices
    [start = 0, num_dimension = 2, global_size = [5,5], local_size = [3,3], local_start = [1,1]] == [6,7,8,11,12,13,16,17,18]

Methods to define a decomposition

- Index list collection
  - Combination of a number of index lists
- Index modifier
  - The modifier allows to define a mapping between indices
  - Modifiers create a new index list from an existing index list
  - Index list: I =[0,1,2,3,4,5]
    Modifier: *M* = [0,2,4,6,8,10]->[10,8,6,4,2,0]
    M(I)=[10,1,8,3,6,5]
- Empty index list
  - …

Generate mapping between src and tgt decomposition

- Exchange maps (Xt_xmap) determine communication partners and what data needs to be exchanged, based on the decomposition
- Algorithms
  1. Every process sends its src and tgt list to all other processes
  2. Rendezvous algorithm[1] based on a distributed directory of global indices is used to avoid all to all communication patterns

[1] *A. Pinar and B. Hendrickson, Communication Support for Adaptive Computation} in Proc. SIAM Conf. on Parallel Processing for Scientific Computing, 2001.*

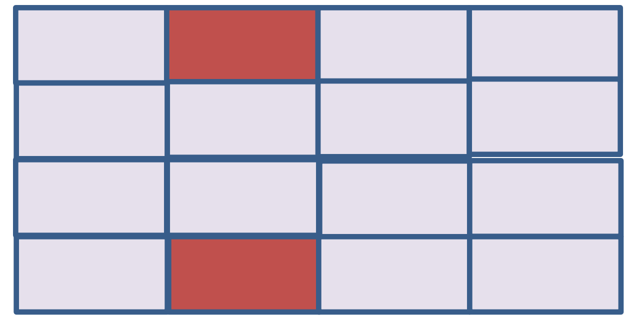
Generate data specific redistribution object

- Redistribution objects (Xt_redist) can be built for every non-null MPI data type (basic types, structs, vectors, …)
- Internally YAXT will build MPI data type for every required exchange → no buffers are required for the exchange
- For a combined redistribution object YAXT will also build MPI data types even if the associated input arrays have no fixed offset between each other

Doing the exchange

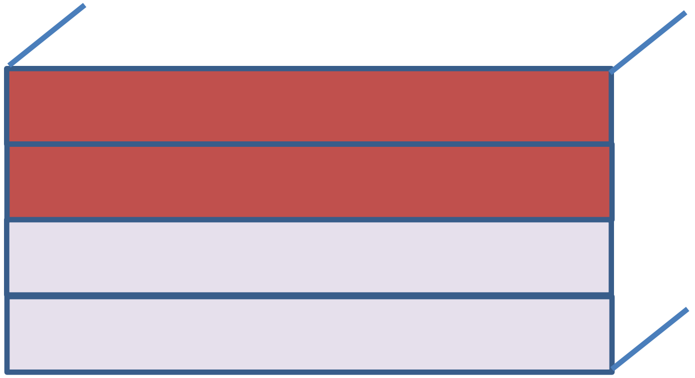
- YAXT currently supports blocking redistribution of data.

Real world example:
ECHAM(gp ←→ ffsl): gp-decomposition



```
TYPE(xt_idxlist) FUNCTION new_gp_3d_idxvec()
    INTEGER :: idx(gp_3d_vol)
    INTEGER :: i, j, k, p, r, index, n
    n = SIZE(idx)
    p = 0
    DO k = 1, nlev
      DO r = 1, 2
        DO j = glats(r), glate(r)
          DO i = glons(r), glone(r)
            index = i + nlon * ( (j-1) + nlat * (k-1) )
            p = p + 1
            idx(p) = index
          ENDDO !i
        ENDDO !j
      ENDDO !r
    ENDDO !k
    new_gp_3d_idxvec = xt_idxvec_new(idx, n)
  END FUNCTION new_gp_3d_idxvec
```

ECHAM(gp ←→ ffsl): ffsl-decomposition

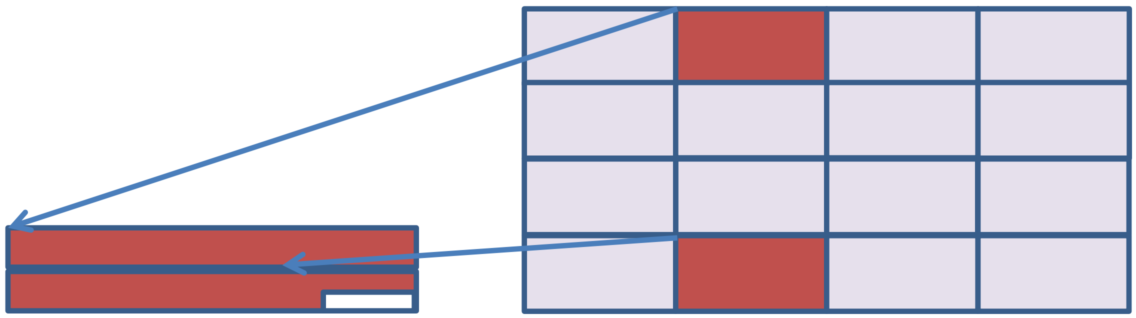


```
TYPE(xt_idxlist) FUNCTION new_ffsl_3d_idxvec()
  INTEGER :: idx(ffsl_3d_vol)
  INTEGER :: i, j, k, kk, p, index, n
  n = size(idx)
  p = 0
  DO kk = 1, ffsl_nlev
    k = ffsl_kstack(kk)
    DO j = ffsl_gp_lat1, ffsl_gp_lat2
      DO i = 1, nlon
        p = p + 1
        index = i + nlon * ( (j-1) + nlat * (k-1) )
        idx(p) = index
      ENDDO
    ENDDO
  ENDDO

  new_ffsl_3d_idxvec = xt_idxvec_new(idx, n)

END FUNCTION new_ffsl_3d_idxvec
```


Jörg Behrens, Moritz Hanke, Thomas Jahns

ECHAM(gp ←→ ffsl): gp-offsets

```
SUBROUTINE set_gp_3d_off(off)
    INTEGER, INTENT(out) :: off(:)
    INTEGER :: coords_off(nproma, nlev, ngpblks)
    INTEGER :: i, j, k, p, r, ib, ia, index, n
    n = SIZE(off)
    p = 0
    DO ib = 1, ngpblks
      DO k = 1, nlev
        DO ia = 1, nproma
          coords_off(ia,k,ib) = p
          p = p + 1
        ENDDO
      ENDDO
    ENDDO
    p = 0
    DO k = 1, nlev
      ib = 1
      ia = 0
      DO r = 1, 2
        DO j = glats(r), glate(r)
          DO i = glons(r), glone(r)
            p = p + 1
            ia = ia +1
            IF (ia > nproma) THEN
              ib = ib + 1
              ia = 1
            ENDIF
            off(p) = coords_off(ia,k,ib)
          ENDDO !i
        ENDDO !j
      ENDDO !r
    ENDDO !k
  END SUBROUTINE set_gp_3d_off
```

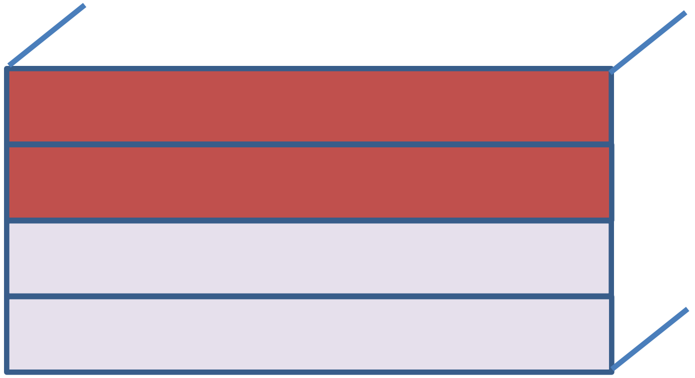


Index to offset relation:
p-th local index corresponds to p-th local offset,
(i,j,k) → p → (ia,k,ib)

Jörg Behrens, Moritz Hanke, Thomas Jahns

ECHAM(gp ←→ ffsl): ffsl-offsets

```
SUBROUTINE set_ffsl_3d_off(off)
   INTEGER, INTENT(out) :: off(:)
   INTEGER :: coords_off(nlon, ffsl_nlat, ffsl_nlev)
   INTEGER :: i, j, k, kk, ic, jc, kc, p, index, n
   n = SIZE(off)
   p = 0
   DO kc = 1, ffsl_nlev
     DO jc = ffsl_nlat, 1, -1 ! change in j-orientation
       DO ic = 1, nlon
         coords_off(ic, jc, kc) = p
         p = p + 1
       ENDDO
     ENDDO
   ENDDO
   p = 0
   DO kk = 1, ffsl_nlev
     kc = kk
     DO j = ffsl_gp_lat1, ffsl_gp_lat2
       jc = j-ffsl_gp_lat1+1
       DO i = 1, nlon
         ic = i
         p = p + 1
         off(p) = coords_off(ic, jc, kc)
       ENDDO
     ENDDO
   ENDDO
 END SUBROUTINE set_ffsl_3d_off
```



Jörg Behrens, Moritz Hanke, Thomas Jahns

ECHAM(gp ←→ ffsl): usage

```
! Definitions:
USE yaxt
TYPE(xt_idxlist) :: gp_3d_idxlist, ffsl_3d_idxlist
TYPE(xt_xmap) :: gp2ffsl_3d_xmap
INTEGER, ALLOCATABLE :: gp_3d_off(:), ffsl_3d_off(:)
TYPE(xt_redist), SAVE :: gp2ffsl_3d_redist

! Decompositions:
gp_3d_idxlist = new_gp_3d_idxvec()
ffsl_3d_idxlist = new_ffsl_3d_idxstripes()

! Init xmap:
gp2ffsl_3d_xmap = xt_xmap_all2all_new(gp_3d_idxlist, ffsl_3d_idxlist,  model_comm)

! Offsets:
CALL set_gp_3d_off(gp_3d_off)
CALL set_ffsl_3d_off(ffsl_3d_off)

! Redist:
gp2ffsl_3d_redist = xt_redist_p2p_off_new(gp2ffsl_3d_xmap, gp_3d_off, ffsl_3d_off, p_real_dp)

! Usage in the model:
CALL xt_redist_s_exchange1(gp2ffsl_3d_redist, C_LOC(gp_data), C_LOC(ffsl_data))
```

Real world example: MPIOM bounds-exchange with modifiers

```
! Different exchange kinds, each of them
! behaves differently at global domain boundaries

INTEGER, PARAMETER :: p_exch_kind      =  1
INTEGER, PARAMETER :: uplus_exch_kind  =  2
INTEGER, PARAMETER :: u_exch_kind      =  3
INTEGER, PARAMETER :: uu_exch_kind     =  4
INTEGER, PARAMETER :: vplus_exch_kind  =  5
INTEGER, PARAMETER :: v_exch_kind      =  6
INTEGER, PARAMETER :: vv_exch_kind     =  7
INTEGER, PARAMETER :: vf_exch_kind     =  8
INTEGER, PARAMETER :: s_exch_kind      =  9
INTEGER, PARAMETER :: sminus_exch_kind = 10

! Usage of yaxt-modifiers to simplify definitions of
! Decompositions
```


Jörg Behrens, Moritz Hanke, Thomas Jahns

Simplified incomplete bounds-exchange example

Global source decomposition

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24

Global target decomposition

5	2	3	4	5	2
11	8	9	10	11	8
17	14	15	16	17	14
23	20	21	22	23	20

Modifier M:

M_1 = [1,7,13,19]→[5,11,17,23]

M_2 = [6,12,18,24]→[2,8,14,20]

$M(I) = M_2(M_1(I))$

Local source indices (including halos):

I_s = [15,16,17,18,21,22,23,24]

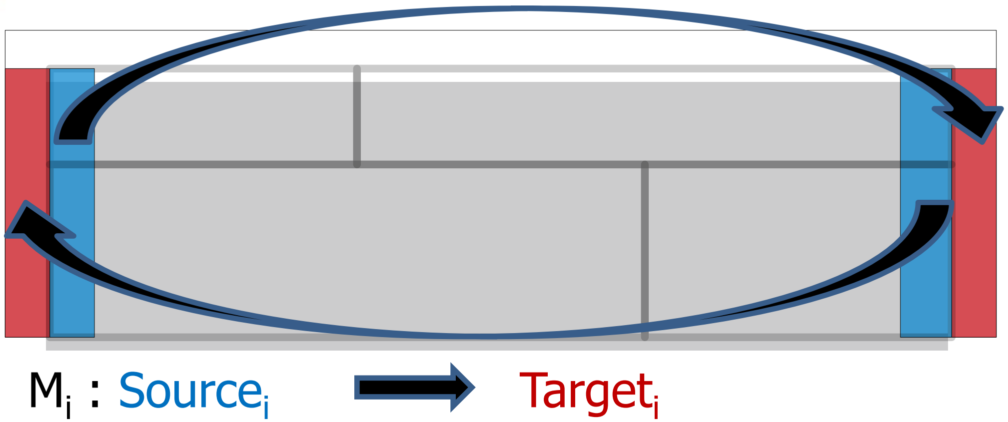
Local target indices (including halos):

$I_t = M(I_s)$ = [15,16,17,14,21,22,23,20]

Modifier can be applied to any index subset

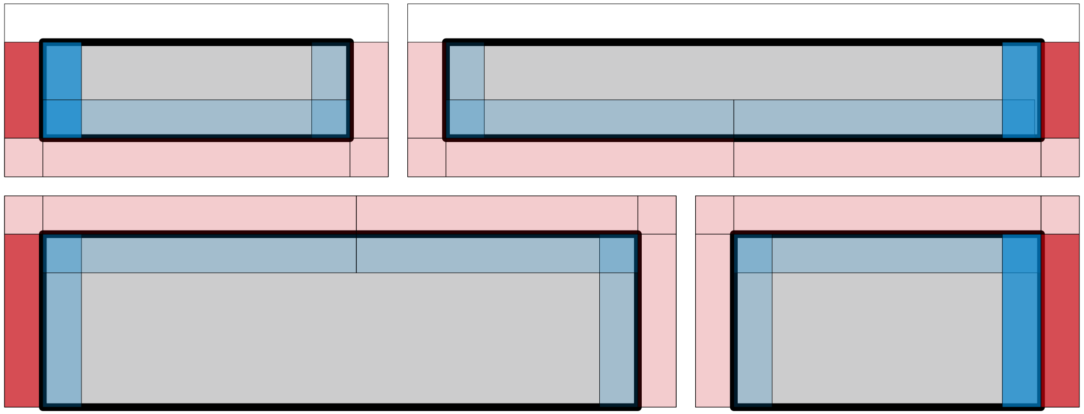
Not included here: Exchange of inner halos

Completing the bounds-exchange example



Global domain halos:

- described by modifiers



Local sub-domain halos:

- Depend on stencil extents
- Local border definition done by user-code

Jörg Behrens, Moritz Hanke, Thomas Jahns

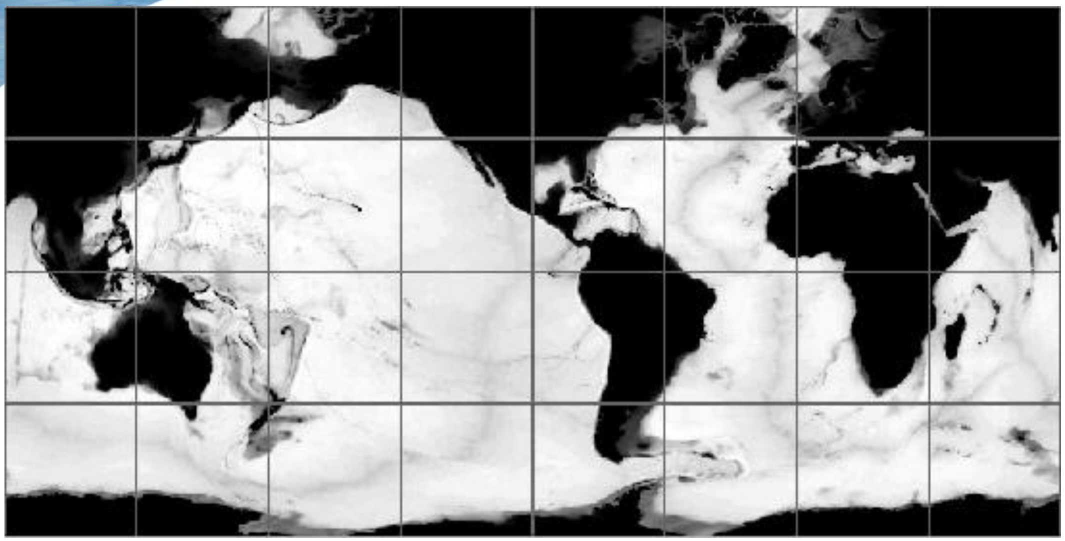
MPIOM 3d-bounds-exchange measurement

Handwritten – requires symmetric regular decomposition

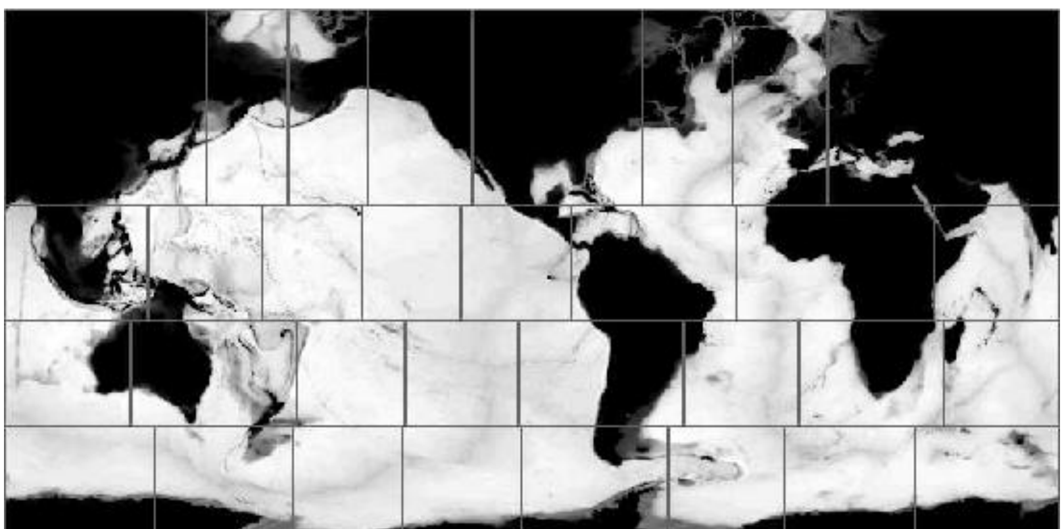
UniTrans/YAXT general solution

Jörg Behrens, Moritz Hanke, Thomas Jahns

MPIOM [TP04L40: 8x4] Load Balance



1. Wet-point-only optimized
   - Workload changed
   - Unfit legacy decomposition
   - Nothing gained



2. Adapted decomposition
   - Old boundary exchange fails
   - Reprogramming exchange?
   - YAXT-formulation:
     - works for both cases
     - faster

Future plans

- Support asynchronous redistributions
- In-Place redistributions
- Multi-Phase exchanges
- Use YAXT in MPIOM, ECHAM, ICON and CDI-PIO

Questions?

Documentation:
https://redmine.dkrz.de/doc/yaxt/html/index.html

Redmine:

https://www.dkrz.de/redmine/projects/yaxt

Download:

https://www.dkrz.de/redmine/projects/yaxt/wiki/Downloads