Reference Manual for Communication Routines for Air Quality Models

Philipp Miehe

Summer 2001
Contents

1 Structures and Terms ........................................... 2
  1.1 File Structure ................................................. 2
  1.2 Module Interrelation .......................................... 3
  1.3 Data Structures ............................................... 3
  1.4 Global Variables ............................................. 4
  1.5 Abbreviations .................................................. 5

2 STEM-III Specific Functions .................................... 6
  2.1 XY-Partitioning ............................................... 6
  2.2 HV-Partitioning ............................................... 13

3 General Communication Functions ................................. 19
  3.1 XY-Partitioning ............................................... 19
  3.2 XY Library .................................................... 22
  3.3 HV-Partitioning ............................................... 32
  3.4 HV Library .................................................... 35
Chapter 1

Structures and Terms

1.1 File Structure

The library is partitioned into a number of files. These files can be divided into two categories: STEM specific files and general files. The STEM specific files can serve as example files to document the necessary function structure that is used for parallelization. Files belonging into this category are: mpi_communication.f, mpi_memalloc, mpi_aq_driver_xy.f and mpi_aq_driver_hv.f.

The rest of the files seen in table 1.1 are general files. As main component of the library, mpi_commlibrary.f encloses the major level of communication routines used in the parallelization. Here routines for each of the data structures used in air quality models are implemented and therefore this file is the main result of our work. The description of the partition mapping and definitions of necessary global variables can be found in mpi_util.f and mpi_util_mpich.f. Here the number of workers for any partitioning is decided on and most of the global variables necessary for parallelization are initialized using the subroutines as described the appendix. The reader is welcome to implement new partitionings, however in order to use the communication library to its full extent the routines have to access and initialize the same global variables, therefore our routines should be used as examples.

The setup files (mpi_xy_setup.f, mpi_hv_setup.f) do not add to the library functionality, but they are necessary to chose the best and fastest implemented communication version for the architecture the AQM is parallelized on. The setup routines time each communication version and write the result into an output file called XY_versions and HV_versions. One of these files is read in during initialization of the associated partition where the communication versions are chosen.

For the different parallelization strategies we use different files. The main programs are started in mpi_aq_driver_xy.f and mpi_aq_driver_hv.f. Both files use the same function calls but different modules since we have set up the modules dependent on the partition. Therefore the XY-partitioning

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpi_xy_setup.f</td>
<td>setup program for XY-partitioning</td>
</tr>
<tr>
<td>mpi_hv_setup.f</td>
<td>setup program for HV-partitioning</td>
</tr>
<tr>
<td>mpi_util.f</td>
<td>communication data types using many MPI-2 functions</td>
</tr>
<tr>
<td>mpi_util_mpich.f</td>
<td>communication data types using functions implemented by MPICH</td>
</tr>
<tr>
<td>mpi_communication.f</td>
<td>STEM-III specific communication functions</td>
</tr>
<tr>
<td>mpi_commlibrary.f</td>
<td>general communication library</td>
</tr>
<tr>
<td>mpi_memalloc.f</td>
<td>general memory allocation functions</td>
</tr>
<tr>
<td>mpi_aq_driver_xy</td>
<td>parallel driver using XY-partitioning</td>
</tr>
<tr>
<td>mpi_aq_driver_hv</td>
<td>parallel driver using HV-partitioning</td>
</tr>
</tbody>
</table>

Table 1.1: List of library files
only calls XY-modules and the HV-partitioning calls HV-modules while both modules may contain
the same function names, however different parameters are used.

In table 1.2 we list the division of the modules into above named files. Note that we have collected
the pairwise associated modules together in one file.

In order to apply a new partitioning one can either introduce a new module or change one of the
given ones to that new partition strategy with respect to the module interrelation as described in
the next section.

The given library functions account for distribution of the necessary arrays as well as data
exchange during time steps and gathering in the end. Usually a few more constants and 1-dimensional
arrays need to be broadcasted as can be seen in mpi_communication.f. Here we implemented STEM-
III specific routines as well as the choice of library routines. Above described library routines can
be hardcode-chosen or runtime-chosen, as the user wishes.

<table>
<thead>
<tr>
<th>Module</th>
<th>File location</th>
</tr>
</thead>
<tbody>
<tr>
<td>XYParallelCommunication</td>
<td>mpi_communication.f</td>
</tr>
<tr>
<td>XYParallelMemAlloc</td>
<td>mpi_memalloc.f</td>
</tr>
<tr>
<td>XYCommDataTypes</td>
<td>mpi_util.f mpi_util_mpich.f</td>
</tr>
<tr>
<td>XYCommunicationLibrary</td>
<td>mpi_commlibrary.f</td>
</tr>
<tr>
<td>XYParallelDataMap</td>
<td>mpi_communication.f</td>
</tr>
<tr>
<td>HVParallelCommunication</td>
<td>mpi_memalloc.f</td>
</tr>
<tr>
<td>HVParallelMemAlloc</td>
<td>mpi_util.f mpi_util_mpich.f</td>
</tr>
<tr>
<td>HVCommDataTypes</td>
<td>mpi_commlibrary.f</td>
</tr>
<tr>
<td>HVCommunicationLibrary</td>
<td>mpi_communication.f</td>
</tr>
<tr>
<td>HVParallelDataMap</td>
<td>mpi_util.f mpi_util_mpich.f</td>
</tr>
</tbody>
</table>

Table 1.2: File location of the modules

1.2 Module Interrelation

From a user perspective only one module needs to be known: the ParallelCommunication. It
includes all the other modules using the FORTRAN use-statement. In order to build a parallelization
only this module has to be included while functions from other modules still can be called.

Figure 1.1 displays the interrelation. By describing the mapping module ParallelDataMap is of
high importance to any routine participating in communication. While module CommDataTypes is
only needed in the actual communication routines given in CommunicationLibrary, ParallelMem-
Alloc is a stand-alone module, its functions can be directly called by the user and are not required
in any other routines.

A few dependencies between modules have to be described. As ParallelDataMap specifies the
partitioning, the global variables initialized here are to be used by the communication routines.
However, we have not been able to use general mappings for the whole communication library. In
CommunicationLibrary only the version 1 (v1-functions) uphold the mapping generality. As can
be seen when looking at the CommDataTypes-module, the communication data structures describe
slices, columns and sets of slices or columns. The latter are implemented in a mapping-dependent
fashion, sets of slices or columns have to be formed generally by combining the data local to one
processor in an abstract way. When using an irregular mapping we lack the ability to construct these
sets to match the communication needs, therefore only the routines sending slice by slice or column
by column (version 1 routines) are able to work for any general mapping, while other routines have
to be ignored.
Figure 1.1: XY-module interrelation

<table>
<thead>
<tr>
<th>Array type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4D</td>
<td>4 dimensional array in dimensions x, y, z and chemical species</td>
</tr>
<tr>
<td>3D</td>
<td>3 dimensional array in dimensions x, y and z</td>
</tr>
<tr>
<td>2DN</td>
<td>3 dimensional array in dimensions x, y and chemical species</td>
</tr>
<tr>
<td>2D</td>
<td>2 dimensional array in dimensions x and y</td>
</tr>
<tr>
<td>BD</td>
<td>4 dimensional boundary array in dimensions x, z, 2 and chemical species or y, z, 2 and chemical species</td>
</tr>
</tbody>
</table>

Table 1.3: Array types representing the grid structure

1.3 Data Structures

An AQM uses special types of data structures. Table 1.3 on the next page lists the array types our library supports.

To describe these structures in more detail we will list a few examples what each of the arrays may be used for. The 4D arrays usually represent the concentration of the chemical species in every grid cell. Therefore it is necessary to distribute and gather these data arrays because in an AQM it will be the data that changes during simulation. 3D arrays may be used for wind components that stay fixed over a certain period of time and will not be computed but delivered as input data, therefore only distribution routines are considered here as well as with any other array mentioned below. 2DN arrays can represent boundary concentration information on the bottom and on top of the grid, therefore only representing x and y dimension. As for the boundary concentration informations on the sides that are represented by BD arrays we have a special structure that combines east and west or north and south boundaries into one array represented by the 2 in the third dimension. And finally 2D arrays can represent data only in the top or bottom layer.

For each of these data structures we provide distribution routines. Furthermore we provide gathering routines as well as shuffling routines between different partitionings in order to make the parallelization work.
<table>
<thead>
<tr>
<th>Variable name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nprocs</td>
<td>number of processors available</td>
</tr>
<tr>
<td>NXworkers</td>
<td>number of workers for X-partitioning</td>
</tr>
<tr>
<td>NYworkers</td>
<td>number of workers for Y-partitioning</td>
</tr>
<tr>
<td>MyId</td>
<td>processor id</td>
</tr>
<tr>
<td>Master</td>
<td>master processor (processor with id 0)</td>
</tr>
<tr>
<td>XWorker</td>
<td>true for all processors with id less or equal NXworkers (excluding the master)</td>
</tr>
<tr>
<td>YWorker</td>
<td>true for all processors with id less or equal NYworkers (excluding the master)</td>
</tr>
</tbody>
</table>

**Table 1.4: Global variables for XY-partitioning**

### 1.4 Global Variables

A major impact on understanding of the main program is based on the global variables used. These are declared in the ParallelDataMap-module, in table 1.4 on the following page we only display the variables for XY-partitioning.

To parallelize an existing AQM program a profound understanding of the partitioning strategy is required. The AQM program includes routines that require a certain layout of the data, this has to be included in the partition considerations. The global variables listed in table 1.4 help to create an environment where partitioning and repartitioning schemes can be applied.

### 1.5 Abbreviations

In the following we explain a few abbreviations used in the appendix.

- no. - number
- calcs - calculations
- dim - dimension
- dir - direction
- conc - concentration
- hor. - horizontal
- ver. - vertical
Chapter 2

STEM-III Specific Functions

This part describes the STEM-III specific functions and their parameters. It can serve as an example on how to implement parallel versions of other AQMs using our library, because understanding of routines gives an insight on how the library functions can be used.

2.1 XY-Partitioning

MemAlloc

Allocates the memory for the arrays specified.

Syntax

```c
MemAlloc(Nx,Ny,Nz,N_gas,N_liq,N_part,
    sg,sl,sp,u,v,w,kh,kv,t,dz,wc,wr,
    sprc,rvel,sx,sy,sz,q,em,vg,fz,hdz,h,tlon,tlat
```

- `INTEGER Nx` ! global no. of grid points in x-dim
- `INTEGER Ny` ! global no. of grid points in y-dim
- `INTEGER Nz` ! global & local no. of grid points in z-dim
- `INTEGER N_gas` ! global no. of gas species dim
- `INTEGER N_liq` ! global no. of liquid species dim
- `INTEGER N_part` ! global no. of particle species dim
- `REAL sg(Nx,Ny,Nz,N_gas)` ! global 4D gas conc array
- `REAL sl(Nx,Ny,Nz,N_liq)` ! global 4D liquid conc array
- `REAL sp(Nx,Ny,Nz,N_part)` ! global 4D particle conc array
- `REAL u(Nx,Ny,Nz)` ! global 3D wind field component array in x-dir
- `REAL v(Nx,Ny,Nz)` ! global 3D wind field component array in y-dir
- `REAL w(Nx,Ny,Nz)` ! global 3D wind field component array in z-dir
- `REAL kh(Nx,Ny,Nz)` ! global 3D hor. diffusivity array
- `REAL kv(Nx,Ny,Nz)` ! global 3D ver. diffusivity array
- `REAL t(Nx,Ny,Nz)` ! global 3D global time array
- `REAL dz(Nx,Ny,Nz)` ! global 3D ver. resolution array
- `REAL wc(Nx,Ny,Nz)` ! global 3D photolysis rate data array
- `REAL wr(Nx,Ny,Nz)` ! global 3D photolysis rate data array
- `REAL sprc(Nx,Ny)` ! global 2D unknown array
- `REAL rvel(Nx,Ny,Nz)` ! global 3D removal velocity array
- `REAL sx(Nx,Nz,2,N_gas)` ! global 2D conc array for east-west boundaries
- `REAL sy(Ny,Nz,2,N_gas)` ! global 2D conc array for north-south boundaries
REAL sz(Nx,Ny,N_gas) ! global 2DN conc array for Z-transport
REAL q(Nx,Ny,N_gas) ! global 2DN unknown array for Z-transport
REAL em(Nx,Ny,Nz,N_gas) ! global 4D emission rates array
REAL vg(Nx,Ny,N_gas) ! global 2D unknown array for Z-transport
REAL fz(Nx,Ny,N_gas) ! global 2D unknown array for input call
REAL hdz(Nx,Ny,Nz) ! global 3D unknown array for chemistry calcs
REAL h(Nx,Ny) ! global 2D top layer height array
REAL tlon(Nx,Ny) ! global 2D unknown array for chemistry calcs
REAL tlat(Nx,Ny) ! global 2D unknown array for chemistry calcs

Details

This routine allocates the memory for all the arrays specified. It can be used to allocate the global as well as the local arrays because it uses the input variables x, y, z and N_gas that can be varied in order to allocate the necessary arrays. Therefore workers can use the routine with specifying local names of the arrays and the local values for x, y, z and N_gas.

Location

mpi_memalloc.f
Module: XYParallelMemAlloc

See Also

MinimAlloc (page 7), MemAlloc for HV-partitioning (page 13)

---

MinimAlloc

Allocates minimal memory for the arrays specified.

Syntax

MinimAlloc(sg,sl,sp,u,v,w,kh,kv,t,dz,wc,wr, sprc,rvel,sx,sy,sz,q,em,vz,hdz,h,tlon,tlad)

REAL sg(1,1,1,1) ! global 4D gas conc array
REAL sl(1,1,1,1) ! global 4D liquid conc array
REAL sp(1,1,1,1) ! global 4D particle conc array
REAL u(1,1,1) ! global 3D wind field component array in x-dir
REAL v(1,1,1) ! global 3D wind field component array in y-dir
REAL w(1,1,1) ! global 3D wind field component array in z-dir
REAL kh(1,1,1) ! global 3D hor. diffusivity array
REAL kv(1,1,1) ! global 3D ver. diffusivity array
REAL t(1,1,1) ! global 3D global time array
REAL dz(1,1,1) ! global 3D ver. resolution array
REAL wc(1,1,1) ! global 3D photolysis rate data array
REAL wr(1,1,1) ! global 3D photolysis rate data array
REAL sprc(1,1) ! global 2D unknown array
REAL rvel(1,1,1) ! global 3D removal velocity array
REAL sx(1,1,2,1) ! global 2D conc array for east-west boundaries
REAL sy(1,1,2,1) ! global 2D conc array for north-south boundaries
REAL sz(1,1,1) ! global 2D conc array for Z-transport
REAL q(1,1,1) ! global 2D unknown array for Z-transport
REAL em(1,1,1,1) ! global 4D emission rates array
REAL vg(1,1,1) ! global 2DN unknown array for Z-transport
REAL fz(1,1,1) ! global 2DN unknown array for input call
REAL hdz(1,1,1) ! global 3D unknown array for chemistry calcs
REAL h(1,1) ! global 2D top layer height array
REAL tlon(1,1) ! global 2D unknown array for chemistry calcs
REAL tlat(1,1) ! global 2D unknown array for chemistry calcs

Details

This routine allocates minimal memory for all the arrays specified. It can be used on the master
to allocate the X and Y-worker specific variables not to be nil pointers as well as on the workers
to minimally allocate the master’s and unnecessary worker specific variables in order to reduce run
time errors with nil pointers.

Location

mpi_memalloc.f
Module: XYParallelMemAlloc

See Also

MemAlloc (page 6), MinimAlloc for HV-partitioning (page 14)

distrib_xy

distribute all arrays required in the model into their X and Y-slice formats.

Syntax

distrib_xy(Nx,Ny,Nz,Nxloc,Nyloc,N_gas,N_liq,N_part,
    sg,sg_x,sg_y,sl_x,sl_y,sp,sp_x,sp_y,
    u,v,w,u_x,v_x,w_x,u_y,v_y,w_y,
    kh,kv,t,kh_x,kv_x,t_x,kh_y,kv_y,t_y,
    wc,wc_x,wc_y,wr,wr_x,wr_y,rvel,rvel_x,rvel_y,
    q,q_x,q_y,prc,prc_x,prc_y,em,em_x,em_y,
    vg,vg_x,vg_y,fz,fz_x,fz_y,hdz,hdz_x,hdz_y,
    sx,sx_y,sy,sy_x,sx_y,sy_y,h,h_x,h_y,
    tlon,tlon_x,tlon_y,tlat,tlat_x,tlat_y,dz,dz_x,dz_y)

INTEGER Nx ! global no. of grid points in x-dim
INTEGER Ny ! global no. of grid points in y-dim
INTEGER Nz ! global & local no. of grid points in z-dim
INTEGER Nxloc ! local no. of grid points in x-dim
INTEGER Nyloc ! local no. of grid points in y-dim
INTEGER N_gas ! local & global no. of gas species dim
INTEGER N_liq ! local & global no. of liquid species dim
INTEGER N_part ! local & global no. of particle species dim
REAL sg(Nx,Ny,Nz,N_gas) ! global 4D gas conc array
REAL sg_x(Nx,Nyloc,Nz,N_gas) ! local 4D gas conc array at X-worker
REAL sg_y(Nxloc,Ny,Nz,N_gas) ! local 4D gas conc array at Y-worker
REAL sl(Nx,Ny,Nz,N_liq) ! global 4D liquid conc array
REAL sl_x(Nx,Nyloc,Nz,N_gas) ! local 4D liquid conc array at X-worker
REAL sl_y(Nxloc,Ny,Nz,N_gas) ! local 4D liquid conc array at Y-worker
REAL sp(Nx,Ny,Nz,Npart) ! global 4D particle conc array
REAL sp_x(Nx,Nyloc,Nz,N_gas) ! local 4D particle conc array at X-worker
REAL sp_y(Nx,Nyloc,Nz,N_gas) ! local 4D particle conc array at Y-worker
REAL u(Nx,Ny,Nz) ! global 3D wind field component array in x-dir
REAL v(Nx,Ny,Nz) ! global 3D wind field component array in y-dir
REAL w(Nx,Ny,Nz) ! global 3D wind field component array in z-dir
REAL u_x(Nx,Nyloc,Nz) ! local 3D wind field component array in x-dir
REAL v_x(Nx,Nyloc,Nz) ! local 3D wind field component array in y-dir
REAL w_x(Nx,Nyloc,Nz) ! local 3D wind field component array in z-dir
REAL u_y(Nxloc,Ny,Nz) ! local 3D wind field component array in z-dir
REAL v_y(Nxloc,Ny,Nz) ! local 3D wind field component array in y-dir
REAL w_y(Nxloc,Ny,Nz) ! local 3D wind field component array in z-dir
REAL kh(Nx,Ny,Nz) ! global 3D hor. diffusitivity array
REAL kv(Nx,Ny,Nz) ! global 3D ver. diffusitivity array
REAL t(Nx,Ny,Nz) ! global 3D global time array
REAL kh_x(Nx,Nyloc,Nz) ! local 3D hor. diffusitivity array at X-worker
REAL kv_x(Nx,Nyloc,Nz) ! local 3D ver. diffusitivity array at X-worker
REAL t_x(Nx,Nyloc,Nz) ! local 3D global time array at X-worker
REAL kh_y(Nxloc,Ny,Nz) ! local 3D hor. diffusitivity array at Y-worker
REAL kv_y(Nxloc,Ny,Nz) ! local 3D ver. diffusitivity array at Y-worker
REAL t_y(Nxloc,Ny,Nz) ! local 3D global time array at Y-worker
REAL wc(Nx,Ny,Nz) ! global 3D photolysis rate data array
REAL wc_x(Nx,Nyloc,Nz) ! local 3D photolysis rate data array at X-worker
REAL wc_y(Nxloc,Ny,Nz) ! local 3D photolysis rate data array at Y-worker
REAL wr(Nx,Ny,Nz) ! global 3D photolysis rate data array
REAL wr_x(Nx,Nyloc,Nz) ! local 3D photolysis rate data array at X-worker
REAL wr_y(Nxloc,Ny,Nz) ! local 3D photolysis rate data array at Y-worker
REAL rvel(Nx,Ny,Nz) ! global 3D removal velocity array
REAL rvel_x(Nx,Nyloc,Nz) ! local 3D removal velocity array at X-worker
REAL rvel_y(Nxloc,Ny,Nz) ! local 3D removal velocity array at Y-worker
REAL q(Nx,Ny,N_gas) ! global 2DN unknown array for Z-transport
REAL q_x(Nx,Nyloc,Nz) ! local 2DN unknown array at X-worker
REAL q_y(Nxloc,Ny,Nz) ! local 2DN unknown array at Y-worker
REAL sprc(Nx,Ny) ! global 2D unknown array
REAL sprc_x(Nx,Nyloc) ! local 2D unknown array at X-worker
REAL sprc_y(Nxloc,Ny) ! local 2D unknown array at Y-worker
REAL em(Nx,Ny,Nz,N_gas) ! global 4D emission rates array
REAL em_x(Nx,Nyloc,Nz,N_gas) ! local 4D emission rates array at X-worker
REAL em_y(Nxloc,Ny,Nz,N_gas) ! local 3D emission rates array at Y-worker
REAL vg(Nx,Ny,N_gas) ! global 2DN deposition velocity array
REAL vg_x(Nx,Nyloc,Nz) ! local 2DN deposition velocity array at X-worker
REAL vg_y(Nxloc,Ny,Nz) ! local 2DN deposition velocity array at Y-worker
REAL fz(Nx,Ny,N_gas) ! global 2DN unknown array for input call
REAL fz_x(Nx,Nyloc,Nz) ! local 2DN unknown array at X-worker
REAL fz_y(Nxloc,Ny,Nz) ! local 2DN unknown array at Y-worker
REAL hdz(Nx,Ny,Nz) ! global 3D unknown array for chemistry calcs
REAL hdz_x(Nx,Nyloc,Nz) ! local 3D unknown array at X-worker
REAL hdz_y(Nxloc,Ny,Nz) ! local 3D unknown array at Y-worker
REAL sx(Ny,Nz,2,N_gas) ! global BD conc array for east-west boundaries
REAL sx(Nyloc,Nz,2,N_gas) ! local BD conc array at X-worker
REAL sy(Nx,Nz,2,N_gas) ! global BD conc array for north-south boundaries
REAL sy(Nxloc,Ny,N_gas) ! local BD conc array at Y-worker
REAL sz(Nx,Ny,N_gas) ! global 2DN conc array for Z-transport
REAL sz(Nx,Nyloc,Nz) ! local 2DN conc array at X-worker
REAL sz_y(Nxloc,Ny,Nz) ! local 2DN conc array at Y-worker
REAL h(Nx,Ny) ! global 2D top layer height array
REAL h_x(Nx,Nyloc) ! local 2D top layer height array at X-worker
REAL h_y(Nxloc,Ny) ! local 2D top layer height array at Y-worker
REAL tlon(Nx,Ny) ! global 2D unknown array for chemistry calcs
REAL tlon_x(Nx,Nyloc) ! local 2D unknown array at X-worker
REAL tlon_y(Nxloc,Ny) ! local 2D unknown array at Y-worker
REAL tlat(Nx,Ny) ! global 2D unknown array for chemistry calcs
REAL tlat_x(Nx,Nyloc) ! local 2D unknown array at X-worker
REAL tlat_y(Nxloc,Ny) ! local 2D unknown array at Y-worker
REAL dz(Nx,Ny,Nz) ! global 3D ver. resolution array
REAL dz_x(Nx,Nyloc,Nz) ! local 3D ver. resolution array at X-worker
REAL dz_y(Nxloc,Ny,Nz) ! local 3D ver. resolution array at Y-worker

Details

All arrays used by the model for X, Y, Z-transport and chemistry calculations are distributed in XY-partitioning using the associated library functions for each array. If further arrays and data are used for the serial model during any of above mentioned computations they will need to be added to this function.

Location

mpi_communication.f
Module: XYParallelCommunication

See Also

distrib_hv (page 15)

---

**int_distrib**

Distribute several integer variables from the master to all workers.

**Syntax**

```
int_distrib(numl,nbin,xtrn,ytrn,ztrn,rxng,rxnl,num,mdt,idade,iend)
```

- INTEGER numl(3,4) !
- INTEGER nbin !
- INTEGER Nxtrn ! decision flag for X-transport
- INTEGER Nytrn ! decision flag for Y-transport
- INTEGER Nztrn ! decision flag for Z-transport
- INTEGER rxng ! decision flag for reaction calcs
- INTEGER rxnl !
- INTEGER num !
- INTEGER mdt ! no. of inner loop iterations
- INTEGER idade(3) ! date of simulation data
- INTEGER iend ! no. of outer loop iterations
Details

This routine distributes several integer variables from the master to all workers by copying the values to a buffer and broadcasting that buffer to all workers. Integer values that may change after the 6-hour interval when new data is read in and therefore need to be redistributed are included in this routine.

Location

mpi_communication.f
Module: XParallelCommunication

See Also

int_distrib for HV-partitioning (page 16)

int_distrib1

Distribute further integer variables from the master to all workers.

Syntax

int_distrib1(ieml,numl)

INTEGER ieml ! no. of outer loop iterations
INTEGER numl(3,4) !

Details

This routine distributes several global known integer variables from the master to all workers by copying the values to a buffer and broadcasting that buffer to all workers. Most of them are index values for certain species that are used very often like water ($H_2O$), air, Oxygen ($O_2$) and Nitrogen ($N_2$) as well as other indices. All variables part of this routine are set in the beginning of the simulation and do not change at any time during computations.

Location

mpi_communication.f
Module: XParallelCommunication

See Also

int_distrib1 for HV-partitioning (page 17)

real_distrib

Distribute several real variables from the master to all workers.

Syntax

real_distrib(Nx,Ny,Nz,Ns,dx,dy,sigmaz,dht,baseh,rmw,dt,ut)
INTEGER Nx ! global no. of grid points in x-dim
INTEGER Ny ! global no. of grid points in y-dim
INTEGER Nz ! global no. of grid points in z-dim
INTEGER Ns ! global no. of chemical species dim
REAL dx(Nx) !
REAL dy(Ny) !
REAL sigmaz(Nz) !
REAL dht !
REAL baseh !
REAL rmw(Ns) !
REAL dt !
REAL ut !

Details
This routine distributes several real variables from the master to all workers by copying the values
to a buffer and broadcasting that buffer to all workers.

Location
mpi_communication.f
Module: XYParallelCommunication

See Also
real_distrib for HV-partitioning (page 18)

---

### cmm_distrib

Distribute all common block data from the master to all workers.

Syntax
cmm_distrib()

Details
This routine distributes all common block data from the master to the workers by copying the values
to several buffers and broadcasting these buffers to all workers.

Location
mpi_communication.f
Module: XYParallelCommunication

See Also
cmm_distrib for HV-partitioning (page 18)
2.2 HV-Partitioning

MemAlloc

Allocates the memory for the arrays specified.

Syntax

```
MemAlloc(Nx,Ny,Nz,N_gas,N_liq,N_part,
          sg,sl,sp,u,v,w,kh,kv,t,dz,wc,wr,
          sprc,rvel,sx,sy,sz,q,em,vf,hz,hdz,tlon,tlat)
```

```
INTEGER Nx           ! global no. of grid points in x-dim
INTEGER Ny           ! global no. of grid points in y-dim
INTEGER Nz           ! global & local no. of grid points in z-dim
INTEGER N_gas        ! global no. of gas species dim
INTEGER N_liq        ! global no. of liquid species dim
INTEGER N_part       ! global no. of particle species dim
REAL sg(Nx,Ny,Nz,N_gas) ! global 4D gas conc array
REAL sl(Nx,Ny,Nz,N_liq) ! global 4D liquid conc array
REAL sp(Nx,Ny,Nz,N_part) ! global 4D particle conc array
REAL u(Nx,Ny,Nz)      ! global 3D wind field component array in x-dir
REAL v(Nx,Ny,Nz)      ! global 3D wind field component array in y-dir
REAL w(Nx,Ny,Nz)      ! global 3D wind field component array in z-dir
REAL kh(Nx,Ny,Nz)     ! global 3D hor. diffusivity array
REAL kv(Nx,Ny,Nz)     ! global 3D ver. diffusivity array
REAL t(Nx,Ny,Nz)      ! global 3D global time array
REAL dz(Nx,Ny,Nz)     ! global 3D ver. resolution array
REAL wc(Nx,Ny,Nz)     ! global 3D photolysis rate data array
REAL wr(Nx,Ny,Nz)     ! global 3D photolysis rate data array
REAL sprc(Nx,Ny)      ! global 2D unknown array
REAL rvel(Nx,Ny,Nz)   ! global 3D removal velocity array
REAL sx(Nx,Ny,2,N_gas) ! global BD conc array for east-west boundaries
REAL sy(Nx,Ny,2,N_gas) ! global BD conc array for north-south boundaries
REAL sz(Nx,Ny,N_gas)  ! global 2DN conc array for Z-transport
REAL q(Nx,Ny,N_gas)   ! global 2DN unknown array for Z-transport
REAL em(Nx,Ny,Nz,N_gas) ! global 4D emission rates array
REAL vg(Nx,Ny,N_gas)  ! global 2DN unknown array for Z-transport
REAL fz(Nx,Ny,N_gas)  ! global 2DN unknown array for input calls
REAL hdz(Nx,Ny,Nz)   ! global 3D unknown array for chemistry calcs
REAL h(Nx,Ny)        ! global 2D top layer height array
REAL tlon(Nx,Ny)     ! global 2D unknown array for chemistry calcs
REAL tlat(Nx,Ny)     ! global 2D unknown array for chemistry calcs
```

Details

This routine allocates the memory for all the arrays specified. It can be used to allocate the global as well as the local arrays because it uses the input variables x, y, z, and \( N_{gas} \) that can be varied in order to allocate the necessary arrays. Therefore workers can use the routine with specifying local names of the arrays and the local values for x, y, z, and \( N_{gas} \).

Location

`mpi_memalloc.f`

Module: HVParallelMemAlloc

13
See Also

MinimAlloc (page 14), MemAlloc for XY-partitioning (page 6)

---

**MinimAlloc**

Allocates minimal memory for the arrays specified.

**Syntax**

```
MinimAlloc (sg, sl, sp, u, v, w, kh, kv, t, dz, wc, wr, sprc, rvel, sx, sy, sz, q, em, vg, fz, hdz, h, tlon, tlad)
```

- `REAL sg(1,1,1)` ! global 4D gas conc array
- `REAL sl(1,1,1)` ! global 4D liquid conc array
- `REAL sp(1,1,1)` ! global 4D particle conc array
- `REAL u(1,1,1)` ! global 3D wind field component array in x-dir
- `REAL v(1,1,1)` ! global 3D wind field component array in y-dir
- `REAL w(1,1,1)` ! global 3D wind field component array in z-dir
- `REAL kv(1,1,1)` ! global 3D ver. diffusitivity array
- `REAL t(1,1,1)` ! global 3D global time array
- `REAL dz(1,1,1)` ! global 3D ver. resolution array
- `REAL wc(1,1,1)` ! global 3D photolysis rate data array
- `REAL wr(1,1,1)` ! global 3D photolysis rate data array
- `REAL sprc(1,1)` ! global 2D unknown array
- `REAL rvel(1,1,1)` ! global 3D removal velocity array
- `REAL sx(1,1,2,1)` ! global 3D conc array for east-west boundaries
- `REAL sy(1,1,2,1)` ! global 3D conc array for north-south boundaries
- `REAL sz(1,1,1)` ! global 2DN conc array for Z-transport
- `REAL q(1,1,1)` ! global 2DN unknown array for Z-transport
- `REAL em(1,1,1,1)` ! global 4D emission rates array
- `REAL vg(1,1,1)` ! global 2DN unknown array for Z-transport
- `REAL fz(1,1,1)` ! global 2DN unknown array for input call
- `REAL hdz(1,1,1)` ! global 3D unknown array for chemistry calcs
- `REAL h(1,1)` ! global 2D top layer height array
- `REAL tlon(1,1)` ! global 2D unknown array for chemistry calcs
- `REAL tlat(1,1)` ! global 2D unknown array for chemistry calcs

**Details**

This routine allocates minimal memory for all the arrays specified. It can be used on the master to allocate the H and V-worker specific variables not to be nil pointers as well as on the workers to allocate the master’s and other worker specific variables in order to reduce run time errors with nil pointers.

**Location**

`mpi_memalloc.f`
Module: HPParallelMemAlloc

See Also
MemAlloc (page 13), MinimAlloc for XY-partitioning (page 7)

distrib_hv

Distribute all arrays required in the model into their H-slice and V-column formats.

Syntax

distrib_hv(Nx,Ny,Nz,Nzloc,NCloc,Ngas,Nliq,Npart,
   sg,sgx,sgy,sl,slx,slx,sp,spx,spz,
   u,v,w,ux,uy,uz,x,y,z,
   th,sv,sw,svx,svy,svz,
   wc,vc,wr,wr,ry,rvx,rvy,rvz,
   q,qx,qy,src,src,src,em,em,em,
   vg,vx,vy,fz,fx,fx,hdz,hdz,
   sx,sy,sz,ex,ez,ez,h,h,h,
   tlon,tlon,tlon,tlat,tlat,tlat,tdz,tdz)

INTEGER Nx ! global no. of grid points in x-dim
INTEGER Ny ! global no. of grid points in y-dim
INTEGER Nz ! global & local no. of grid points in z-dim
INTEGER Nzloc ! local no. of grid points in z-dim
INTEGER NCloc ! local no. of ver. columns
INTEGER Ngas ! local & global no. of gas species dim
INTEGER Nliq ! local & global no. of liquid species dim
INTEGER Npart ! local & global no. of particle species dim
REAL sg(Nx,Ny,Nz,Ngas) ! global 4D gas conc array
REAL sgx(Nx,Ny,Nzloc,Ngas) ! local 4D gas conc array at H-worker
REAL sgy(Nx,Nzloc,Ngas) ! local 4D gas conc array at V-worker
REAL sl(Nx,Ny,Nz,Nliq) ! global 4D liquid conc array
REAL slx(Nx,Ny,Nzloc,Ngas) ! local 4D liquid conc array at H-worker
REAL slz(Nx,Nzloc,Ngas) ! local 4D liquid conc array at V-worker
REAL sp(Nx,Ny,Nz,Npart) ! global 4D particle conc array
REAL spx(Nx,Ny,Nzloc,Ngas) ! local 4D particle conc array at H-worker
REAL spy(Nx,Nzloc,Ngas) ! local 4D particle conc array at V-worker
REAL u(Nx,Ny,Nz) ! global 3D wind field component array in x-dir
REAL v(Nx,Ny,Nz) ! global 3D wind field component array in y-dir
REAL w(Nx,Ny,Nz) ! global 3D wind field component array in z-dir
REAL uh(Nx,Ny,Nzloc) ! local 3D wind field component array in x-dir
   at H-worker
REAL vh(Nx,Ny,Nzloc) ! local 3D wind field component array in y-dir
   at H-worker
REAL wv(Nx,Nzloc,Nz) ! local 3D wind field component array in z-dir
   at H-worker
REAL kh(Nx,Ny,Nz) ! global 3D hor. diffusivity array
REAL kv(Nx,Ny,Nz) ! global 3D ver. diffusivity array
REAL t(Nx,Ny,Nz) ! global 3D global time array
REAL khh(Nx,Ny,Nzloc) ! local 3D hor. diffusivity array at H-worker
REAL kvv(Nx,Nzloc,Nz) ! local 3D global time array at H-worker
REAL wc(Nx,Ny,Nz) ! global 3D global time array at V-worker
REAL wcv(Nx,Nzloc,Nz) ! local 3D photolysis rate data array at V-worker
REAL wr(Nx,Ny,Nz) ! global 3D photolysis rate data array at V-worker
REAL wrny(1,Ncloc,Nz) ! local 3D photolysis rate data array at V-worker
REAL rvel(Nx,Ny,Nz) ! global 3D removal velocity array
REAL rvelny(1,Ncloc,Nz) ! local 3D removal velocity array at V-worker
REAL q(Nx,Ny,N_gas) ! global 2DN unknown array for Z-transport
REAL qny(1,Ncloc,Nz) ! local 2DN unknown array at V-worker
REAL sprc(Nx,Ny) ! global 2D unknown array
REAL sprcny(1,Ncloc) ! local 2D unknown array at V-worker
REAL em(Nx,Ny,Nz,N_gas) ! global 4D emission rates array
REAL emny(1,Ncloc,Nz,N_gas) ! local 3D emission rates array at V-worker
REAL vg(Nx,Ny,N_gas) ! global 2DN deposition velocity array
REAL vgy(1,Ncloc,Nz) ! local 2DN deposition velocity array at V-worker
REAL fz(Nx,Ny,N_gas) ! global 2DN unknown array for input call
REAL fzny(1,Ncloc,Nz) ! local 2DN unknown array at V-worker
REAL hdz(Nx,Ny,Nz) ! global 3D unknown array for chemistry calc's
REAL hdzny(1,Ncloc,Nz) ! local 3D unknown array at V-worker
REAL sx(Nx,Ny,2,N_gas) ! global BD conc array for east-west boundaries
REAL sxny(1,Ncloc,2,N_gas) ! local BD conc array at H-worker
REAL sy(Ny,Nz,2,N_gas) ! global BD conc array for north-south boundaries
REAL syny(1,Ncloc,2,N_gas) ! local BD conc array at H-worker
REAL sz(Nx,Ny,N_gas) ! global 2DN conc array for Z-transport
REAL szny(1,Ncloc,N_gas) ! local 2DN conc array at V-worker
REAL h(Nx,Ny) ! global 2D top layer height array
REAL hny(1,Ncloc) ! local 2D top layer height array at V-worker
REAL tlon(Nx,Ny) ! global 2D unknown array for chemistry calc's
REAL tlonny(1,Ncloc) ! local 2D unknown array at V-worker
REAL tlat(Nx,Ny) ! global 2D unknown array for chemistry calc's
REAL tlatny(1,Ncloc) ! local 2D unknown array at V-worker
REAL dz(Nx,Ny,Nz) ! global 3D ver. resolution array
REAL dzny(1,Ncloc,Nz) ! local 3D ver. resolution array at V-worker

Details

All arrays used by the model for X, Y, X-transport and chemistry calculations are distributed in HV-partitioning using the associated library functions for each array. If further arrays and data are used for the serial model during any of above mentioned computations they will need to be added to this function.

Location

mpi_communication.f
Module: HVParallelCommunication

See Also
distrib_xy (page 8)

int_distrib

Distribute several integer variables from the master to all workers.

Syntax

int_distrib(numl,nbin,xtrn,ytrn,ztrn,rxng,rxnl,num,mdt,idate,iend)
INTEGER numl(3,4)        !
INTEGER nbin            !
INTEGER Nxtrn          ! decision flag for X-transport
INTEGER Nytrn          ! decision flag for Y-transport
INTEGER Nztrn          ! decision flag for Z-transport
INTEGER rxmg           ! decision flag for reaction calcs
INTEGER rxn1           !
INTEGER num             !
INTEGER mdt            ! no. of inner loop iterations
INTEGER idate(3)       ! date of simulation data
INTEGER iend            ! no. of outer loop iterations

Details

This routine distributes several integer variables from the master to all workers by copying the values to a buffer and broadcasting that buffer to all workers. Integer values that may change after the 6-hour interval when new data is read in and therefore need to be redistributed are included in this routine.

Location

mpi_communication.f
Module: HVParallelCommunication

See Also

int_distrib for XY-partitioning (page 10)

---

**int_distrib1**

Distribute further integer variables from the master to all workers.

Syntax

int_distrib1(iend,numl)

    INTEGER iend            ! no. of outer loop iterations
    INTEGER numl(3,4)        !

Details

This routine distributes several global known integer variables from the master to all workers by copying the values to a buffer and broadcasting that buffer to all workers. All variables part of this routine are set in the beginning of the simulation and do not change at any time during computations.

Location

mpi_communication.f
Module: HVParallelCommunication

See Also
real_distrib

Distribute several real variables from the master to all workers.

Syntax

```
real_distrib(Nx,Ny,Nz,Ns,dx,dy,sigmax,dht,baseh,rmw,dt,ut)
```

- `INTEGER Nx` ! global no. of grid points in x-dim
- `INTEGER Ny` ! global no. of grid points in y-dim
- `INTEGER Nz` ! global no. of grid points in z-dim
- `INTEGER Ns` ! global no. of chemical species dim
- `REAL dx(Nx)` !
- `REAL dy(Ny)` !
- `REAL sigmax(Nz)` !
- `REAL dht` !
- `REAL baseh` !
- `REAL rmw(Ns)` !
- `REAL dt` !
- `REAL ut` !

Details

This routine distributes several real variables from the master to all workers by copying the values to a buffer and broadcasting that buffer to all workers.

Location

`mpi_communication.f`
Module: HVParallelCommunication

See Also

real_distrib for XY-partitioning (page 11)

cmm_distrib

Distribute all common block data from the master to all workers.

Syntax

```
cmm_distrib()
```

Details

This routine distributes all common block data from the master to the workers by copying the values to several buffers and broadcasting these buffer to all workers.

Location

`mpi_communication.f`
Module: HVParallelCommunication

See Also

cmm_distrib for XY-partitioning (page 12)
Chapter 3

General Communication Functions

The following general routines need to be included in order to use the communication library. While functions described in the last chapter were STEM-III specific, routines illustrated in this chapter are the general basis of the library.

3.1 XY-Partitioning

init_xy

Initialize and set up the communication for XY-partitioning.

Syntax

init_xy(Nx,Ny,Nz,Nxloc,Nyloc,Ns)

- INTEGER Nx  ! global no. of grid points in x-dim
- INTEGER Ny  ! global no. of grid points in y-dim
- INTEGER Nz  ! global no. of grid points in z-dim
- INTEGER Nxloc  ! local no. of grid points in x-dim on Y-workers
- INTEGER Nyloc  ! local no. of grid points in y-dim on X-workers
- INTEGER Ns  ! no. of chemical species to be monitored

Details

This routine initializes the parallel communication for XY-partitioning by calling the three later described functions init_xy_versions, CreateMap and CreateCommDataTypes. Calling this routine instead of the others makes sure that the right modules are included.

Location

mpi_communication.f
Module: XYParallelCommunication

See Also

init_xy_versions (page 20), CreateMap (page 19), CreateCommDataTypes (page 21)

CreateMap

Create the mapping between X and Y-slices and processors as well as initializing global variables.
Syntax

CreateMap(Nx,Ny,Nxloc,Nyloc)

| INTEGER Nx | ! global no. of grid points in x-dim |
| INTEGER Ny | ! global no. of grid points in y-dim |
| INTEGER Nxloc | ! local no. of grid points in x-dim on Y-workers |
| INTEGER Nyloc | ! local no. of grid points in y-dim on X-workers |

Details

This routine is part of the module ParallelDataMap. It assigns the processors to be X-workers and Y-workers, respectively and keeps book about the assignments of the global X and Y-slices to these workers. This includes the number of X and Y-slices assigned to each worker, their global and local id, as well as the owner relationship to each X and Y-slice.

Most global variables are set in this routine: the number of X and Y-workers, the size of the local arrays at the workers as well as mapping variables. The provided routine can serve as an example on how to set up a different mapping. We have used the following global mapping variables:

| owner_of_xslice(1..Ny) | specifies the process that gets the X-slice |
| owner_of_yslice(1..Nx) | specifies the process that gets the Y-slice |
| no_of_xslices(p) | number of X-slices owned by process p where \( p \in \{1,\ldots,\text{NXworkers}\} \) |
| no_of_yslices(p) | number of Y-slices owned by process p where \( p \in \{1,\ldots,\text{NYworkers}\} \) |
| local_xslice_id(1..Ny) | local index of the global X-slice (index seen by the process which owns it) |
| local_yslice_id(1..Nx) | local index of the global Y-slice (index seen by the process which owns it) |
| global_xslice_id(p,1..Nyloc) | global index of a local X-slice on processor p |
| global_yslice_id(p,1..Nxloc) | global index of a local Y-slice on processor p |
| owned_xslices(p,1..no_of_xslices(p)) | global X-slices owned by worker p |
| owned_yslices(p,1..no_of_yslices(p)) | global Y-slices owned by worker p |

Location

mpi_util.f
Module: XYParallelDataMap

See Also

CreateMap for HV-partitioning (page 33)

---

init_xy_versions

Initialize the communication versions to be used.

Syntax

init_xy_versions()
Details

This routine reads the file created by the setup program that specifies the fastest version numbers for the communication routines on the specific architecture using a set grid resolution and given number of processors. If the setup program has not been run the default version (version 1) will be used.

To decide on the versions during runtime either a few subroutines have to be provided that map the integers specified by the setup program (1, 2, 3) to the communication routine version. Another approach is hardcoding the best routines for the used architecture in the AQM-specific functions after testing the speed of each version using the setup programs.

Location

mpi_communication.f
Module: XYParallelCommunication

---

CreateCommDataTypes

Creates the data types used by the library routines for communication.

Syntax

CreateCommDataTypes(Nx,Ny,Nz,Nxloc,Nyloc,Ns)

| INTEGER Nx     | ! global no. of grid points in x-dim |
| INTEGER Ny     | ! global no. of grid points in y-dim |
| INTEGER Nxloc | ! local no. of grid points in x-dim on Y-workers |
| INTEGER Nyloc | ! local no. of grid points in y-dim on X-workers |
| INTEGER Nz     | ! global no. of grid points in z-dim |
| INTEGER Ns     | ! no. of chemical species to be monitored |

Details

This routine creates the worker group used for communication during shuffling as well as the communication data types for each of the array types in the air quality model. There have to be slice-types to do the slice-communication in distribution and gathering. Other data types created are the combined slices for the single message distribution and gathering as well as the combined data to be sent during shuffling when using one message only. These data types represent parts of an array that can be located anywhere in the array specified by count numbers and displacements.

Most of these data types depend on the mapping function described previously. The implementation can serve as an example on how to define these data types when using different mapping routines. Be aware that the combining of X and Y-slices to one data type for single message communication may not be possible if the mapping is irregular! In that case only the first version of communication routines can be used that distribute and gather the X and Y-slices slice by slice.

Location

mpi_util.f
Module: XYCommDataTypes

See Also

CreateCommDataTypes for HV-partitioning (page 34)
3.2 XY Library

\texttt{distrib.x.4D}

Distribute a 4D array in X-slice partitioning from the master to the X-workers.

Syntax
\begin{verbatim}
distrib.x.4D(x|y|z|3)(Nx,Ny,Nz,Nyloc,Ns,aglob,alloc)
\end{verbatim}

- INTEGER \texttt{Nx} ! global & local no. of grid points in x-dim
- INTEGER \texttt{Ny} ! global & local no. of grid points in y-dim
- INTEGER \texttt{Nz} ! global & local no. of grid points in z-dim
- INTEGER \texttt{Nyloc} ! local no. of grid points in y-dim
- INTEGER \texttt{Ns} ! local & global no. of chemical species dim
- REAL \texttt{aglob(Nx,Ny,Nz,Ns)} ! global 4D array
- REAL \texttt{alloc(Nx,Nyloc,Nz,Ns)} ! local 4D array on each X-worker

Details
These routines distribute a global 4D array at the master into X-slice partitioning to the workers. The following three versions are implemented:

\begin{itemize}
  \item \texttt{v1} - distribution sending slice by slice using \texttt{Ny} messages
  \item \texttt{v2} - distribution sending one message to each X-worker combining the slices if more than one
  \item \texttt{v3} - using two calls to \texttt{MPI\_SCATTERV}
\end{itemize}

Note that the first routine is independent of the mapping while the second and third routines depend on the data types declared for a certain mapping in the function \texttt{CreateCommDatatypes} and therefore can not be used with any other mapping unless the communication structures have been reimplemented.

Location
\texttt{mpi\_commlibrary.f}
Module: XYCommunicationLibrary

See Also
\texttt{distrib.y.4D (page 22)}, \texttt{distrib.h.4D (page 35)}

\texttt{distrib.y.4D}

Distribute a 4D array in Y-slice partitioning from the master to the Y-workers.

Syntax
\begin{verbatim}
distrib.y.4D(x|y|z|3)(Nx,Ny,Nz,Nxloc,Ns,aglob,alloc)
\end{verbatim}

- INTEGER \texttt{Nx} ! global no. of grid points in x-dim
- INTEGER \texttt{Ny} ! global & local no. of grid points in y-dim
- INTEGER \texttt{Nz} ! global & local no. of grid points in z-dim
- INTEGER \texttt{Nxloc} ! local no. of grid points in x-dim
- INTEGER \texttt{Ns} ! local & global no. of chemical species dim
- REAL \texttt{aglob(Nx,Ny,Nz,Ns)} ! global 4D array
- REAL \texttt{alloc(Nxloc,Ny,Nz,Ns)} ! local 4D array on each Y-worker
Details

These routines distribute a global 4D array at the master into Y-slice partitioning to the workers. The following three versions are implemented:

- v1 - distribution sending slice by slice using x messages
- v2 - distribution sending one message to each Y-worker combining the slices if more than one
- v3 - using two calls to MPI_SCATTERV

Note that the first routine is independent of the mapping while the second and third routines depend on the data types declared for a certain mapping in the function CreateCommDatatypes and therefore can not be used with any other mapping unless the communication structures have been reimplemented.

Location

mpi_commLibrary.f
Module: XYCommunicationLibrary

See Also
distrib_x_4D (page 22), distrib_y_4D (page 35)

gather_x_4D

Gather a 4D array at the master from X-slice partitioning at the X-workers.

Syntax

gather_x_4D,v[1|2|3](Nx,Ny,Nz,Nyloc,Ns,aglob,aloc)

<table>
<thead>
<tr>
<th>INTEGER Nx</th>
<th>! global &amp; local no. of grid points in x-dim</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEGER Ny</td>
<td>! global no. of grid points in y-dim</td>
</tr>
<tr>
<td>INTEGER Nz</td>
<td>! global &amp; local no. of grid points in z-dim</td>
</tr>
<tr>
<td>INTEGER Nyloc</td>
<td>! local no. of grid points in y-dim</td>
</tr>
<tr>
<td>INTEGER Ns</td>
<td>! local &amp; global no. of chemical species dim</td>
</tr>
<tr>
<td>REAL aglob(Nx,Ny,Nz,Ns)</td>
<td>! global 4D array</td>
</tr>
<tr>
<td>REAL aloc(Nx,Nyloc,Nz,Ns)</td>
<td>! local 4D array on each X-worker</td>
</tr>
</tbody>
</table>

Details

These routines gather the local 4D arrays at the X-workers into the global 4D array at the master. The following three versions are implemented:

- v1 - master gathering, receiving slice by slice in Ny messages
- v2 - master gathering, receiving one message from each X-worker combining the local slices if more than one
- v3 - using two calls to MPI_GATHERV

Note that the first routine is independent of the mapping while the second and third routines depend on the data types declared for a certain mapping in the function CreateCommDatatypes and therefore can not be used with any other mapping unless the communication structures have been reimplemented.
Location

mpi_commmlibrary.f
Module: XYCommunicationLibrary

See Also

gather_y_4D (page 24), gather_h_4D (page 36)

gather_y_4D

Gather a 4D array at the master from Y-slice partitioning at the Y-workers.

Syntax

gather_y_4D[n|2|3](Nx,Ny,Nz,Nxloc,Ns,aglob,aloc)

INTEGER Nx  ! global no. of grid points in x-dim
INTEGER Ny  ! global & local no. of grid points in y-dim
INTEGER Nz  ! global & local no. of grid points in z-dim
INTEGER Nxloc ! local no. of grid points in x-dim
INTEGER Ns  ! local & global no. of chemical species dim
REAL aglob(Nx,Ny,Nz,Ns) ! global 4D array
REAL aloc(Nxloc,Ny,Nz,Ns) ! local 4D array on each Y-worker

Details

These routines gather the local 4D arrays at the Y-workers into the global 4D array at the master. The following three versions are implemented:

v1 - master gathering, receiving slice by slice in x messages
v2 - master gathering, receiving one message from each Y-worker combining the local slices if more than one
v3 - using two calls to MPI_GATHERV

Note that the first routine is independent of the mapping while the second and third routines depend on the data types declared for a certain mapping in the function CreateCommDatatypes and therefore cannot be used with any other mapping unless the communication structures have been reimplemented.

Location

mpi_commmlibrary.f
Module: XYCommunicationLibrary

See Also

gather_x_4D (page 23), gather_y_4D (page 37)

shuffle_x2y_4D

Shuffle 4D arrays from X-partitioning on X-workers to Y-partitioning on Y-workers.

Syntax
shuffle_x2y_4D(Nx,Ny,Nz,Nxloc,Nyloc,Ns,alocx,alocy)

INTEGER Nx ! global no. of grid points in x-dim
INTEGER Ny ! global no. of grid points in y-dim
INTEGER Nz ! global & local no. of grid points in z-dim
INTEGER Nxloc ! local no. of grid points in x-dim
INTEGER Nyloc ! local no. of grid points in y-dim
INTEGER Ns  ! local & global no. of chemical species dim
REAL alocx(Nx,Nyloc,Nz,Ns) ! local 4D array on each X-worker
REAL alocy(Nxloc,Ny,Nz,Ns) ! local 4D array on each Y-worker

Details

These routines rearrange the local 4D in X-partitioning at the X-workers to Y-partitioning at the Y-workers. The following three versions are implemented:

v1 - distribution sending slice-part by slice-part using Nyloc messages on each X-worker

v2 - distribution sending one message from each X-worker to each Y-worker combining the necessary data

v3 - using copying and MPI_ALLTOALL

Note that the first routine is independent of the mapping while the second and third routines depend on the data types declared for a certain mapping in the function CreateCommDatatypes and therefore can not be used with any other mapping unless the communication structures have been reimplemented.

Location

mpi_commlibrary.f
Module: XYCommunicationLibrary shuffle_x2y_4D (page 25), shuffle_h2v_4D (page 37)

shuffle_y2x_4D

Shuffle 4D arrays from Y-partitioning on Y-workers to X-partitioning on X-workers.

Syntax

shuffle_y2x_4D(Nx,Ny,Nz,Nxloc,Nyloc,Ns,alocx,alocy)

INTEGER Nx ! global no. of grid points in x-dim
INTEGER Ny ! global no. of grid points in y-dim
INTEGER Nz ! global & local no. of grid points in z-dim
INTEGER Nxloc ! local no. of grid points in x-dim
INTEGER Nyloc ! local no. of grid points in y-dim
INTEGER Ns  ! local & global no. of chemical species dim
REAL alocx(Nx,Nyloc,Nz,Ns) ! local 4D array on each X-worker
REAL alocy(Nxloc,Ny,Nz,Ns) ! local 4D array on each Y-worker

Details

These routines rearrange the local 4D in Y-partitioning at the Y-workers to X-partitioning at the X-workers. The following three versions are implemented:

v1 - distribution sending slice-part by slice-part using Nxloc messages on each Y-worker
v2 - distribution sending one message from each Y-worker to each X-worker combining the necessary data

v3 - using copying and MPI_ALLTOALL

Note that the first routine is independent of the mapping while the second and third routines depend on the data types declared for a certain mapping in the function CreateCommDatatypes and therefore can not be used with any other mapping unless the communication structures have been reimplemented.

Location

mpi_commlibrary.f
Module: XYCommunicationLibrary shuffle_x2y_4D (page 24), shuffle_y2h_4D (page 38)

---

distrib_x_2D

Distribute a 2D array in X-slice partitioning from the master to the X-workers.

Syntax

distrib_x_2D_Nx,Ny,Nxloc,aglob,alloc)

| INTEGER Nx | ! global & local no. of grid points in x-dim |
| INTEGER Ny | ! global no. of grid points in y-dim |
| INTEGER Nyloc | ! local no. of grid points in y-dim |
| REAL aglob(Nx,Ny) | ! global 2D array |
| REAL alloc(Nx,Nyloc) | ! local 2D array on each X-worker |

Details

These routines distribute a global 2D array at the master into X-slice partitioning to the workers. The following three versions are implemented:

v1 - distribution sending slice by slice using Ny messages

v2 - distribution sending one message to each X-worker combining the slices if more than one

v3 - using two calls to MPI_SCATTERV

Note that the first routine is independent of the mapping while the second and third routines depend on the data types declared for a certain mapping in the function CreateCommDatatypes and can not be used with any other mapping unless the communication structures have been reimplemented.

Location

mpi_commlibrary.f
Module: XYCommunicationLibrary distrib_y_2D (page 26), distrib_h_2D (page 39)

---

distrib_y_2D

Distribute a 2D array in Y-slice partitioning from the master to the Y-workers.

Syntax

distrib_y_2D_Nx,Ny,Nxloc,aglob,alloc)
INTEGER Nx ! global no. of grid points in x-dim
INTEGER Ny ! global & local no. of grid points in y-dim
INTEGER Nxloc ! local no. of grid points in x-dim
REAL aglob(Nx,Ny) ! global 2D array
REAL aloc(Nxloc,Ny) ! local 2D array on each Y-worker

Details

These routines distribute a global 2D array at the master into Y-slice partitioning to the workers. The following three versions are implemented:

v1 - distribution sending slice by slice using Nx messages

v2 - distribution sending one message to each Y-worker combining the slices if more than one

v3 - using two calls to MPI_SCATTERV

Note that the first routine is independent of the mapping while the second and third routines depend on the data types declared for a certain mapping in the function CreateCommDatatypes and therefore can not be used with any other mapping unless the communication structures have been reimplemented.

Location

mpi_commlibrary.f
Module: XYCommunicationLibrary distrib_x_2D (page 26), distrib_y_2D (page 39)

distrib_x_2DN

Distribute a 2DN array in X-slice partitioning from the master to the X-workers.

Syntax

distrib_x_2DN,\nu[1|2|3](Nx,Ny,Nyloc,Ns,aglob,aloc)

INTEGER Nx ! global & local no. of grid points in x-dim
INTEGER Ny ! global no. of grid points in y-dim
INTEGER Nyloc ! local no. of grid points in y-dim
INTEGER Ns ! local & global no. of chemical species dim
REAL aglob(Nx,Ny,Ns) ! global 2DN array
REAL aloc(Nx,Nyloc,Ns) ! local 2DN array on each X-worker

Details

These routines distribute a global 2DN array at the master into X-slice partitioning to the workers. A 2DN array is a 3D array in the dimensions x, y, and number of chemical species. The following three versions are implemented:

v1 - distribution sending slice by slice using Ny messages

v2 - distribution sending one message to each X-worker combining the slices if more than one

v3 - using two calls to MPI_SCATTERV

Note that the first routine is independent of the mapping while the second and third routines depend on the data types declared for a certain mapping in the function CreateCommDatatypes and therefore can not be used with any other mapping unless the communication structures have been reimplemented.
Location

mpi_commlibrary.f
Module: XYCommunicationLibrary distrib_y_2DN (page 28)

---

**distrib_y_2DN**

Distribute a 2DN array in Y-slice partitioning from the master to the Y-workers.

**Syntax**

```fortran
   distrib_y_2DN_y[1|2|3](Nx,Ny,Nxloc,Ns,aglob,aloc)
   INTEGER Nx        ! global no. of grid points in x-dim
   INTEGER Ny        ! global & local no. of grid points in y-dim
   INTEGER Nxloc     ! local no. of grid points in x-dim
   INTEGER Ns        ! local & global no. of chemical species dim
   REAL aglob(Nx,Ny,Ns)  ! global 2DN array
   REAL aloc(Nxloc,Ny,Ns) ! local 2DN array on each Y-worker
```

**Details**

These routines distribute a global 2DN array at the master into Y-slice partitioning to the workers. A 2DN array is a 3D array in the dimensions x, y, and number of chemical species. The following three versions are implemented:

- **v1** - distribution sending slice by slice using Nx messages
- **v2** - distribution sending one message to each Y-worker combining the slices if more than one
- **v3** - using two calls to MPI_SCATTERV

Note that the first routine is independent of the mapping while the second and third routines depend on the data types declared for a certain mapping in the function CreateCommDatatypes and therefore cannot be used with any other mapping unless the communication structures have been reimplemented.

Location

mpi_commlibrary.f
Module: XYCommunicationLibrary distrib_x_2DN (page 27), distrib_y_2DN (page 40)

---

**distrib_x_3D**

Distribute a 3D array in X-slice partitioning from the master to the X-workers.

**Syntax**

```fortran
   distrib_x_3D_y[1|2|3](Nx,Ny,Nz,Nyloc,aglob,aloc)
   INTEGER Nx            ! global & local no. of grid points in x-dim
   INTEGER Ny            ! global no. of grid points in y-dim
   INTEGER Nz            ! global & local no. of grid points in z-dim
   INTEGER Nyloc         ! local no. of grid points in y-dim
   REAL aglob(Nx,Ny,Nz)   ! global 3DN array
   REAL aloc(Nx,Nyloc,Nz) ! local 3DN array on each X-worker
```
Details

These routines distribute a global 3D array at the master into X-slice partitioning to the workers. The following three versions are implemented:

v1 - distribution sending slice by slice using Ny messages
v2 - distribution sending one message to each X-worker combining the slices if more than one
v3 - using two calls to MPI_SCATTERV

Note that the first routine is independent of the mapping while the second and third routines depend on the data types declared for a certain mapping in the function CreateCommDatatypes and therefore can not be used with any other mapping unless the communication structures have been reimplemented.

Location

mpi_commlibrary.f
Module: XYCommunicationLibrary distrib_y_3D (page 29), distrib_z_3D (page 41)

---

distrib_y_3D

Distribute a 3D array in Y-slice partitioning from the master to the Y-workers.

Syntax

distrib_y_3D[y|z] (Nx,Ny,Nz,Nxloc,aglob,alloc)

integer nx          ! global no. of grid points in x-dim
integer ny          ! global & local no. of grid points in y-dim
integer nz          ! global & local no. of grid points in z-dim
real aglob(nx,ny,nz) ! global 3D array
real alloc(nxloc,ny,nz) ! local 3D array on each Y-worker

Details

These routines distribute a global 3D array at the master into Y-slice partitioning to the workers. The following three versions are implemented:

v1 - distribution sending slice by slice using Nx messages
v2 - distribution sending one message to each Y-worker combining the slices if more than one
v3 - using two calls to MPI_SCATTERV

Note that the first routine is independent of the mapping while the second and third routines depend on the data types declared for a certain mapping in the function CreateCommDatatypes and therefore can not be used with any other mapping unless the communication structures have been reimplemented.

Location
**distrib\_x\_BD**

Distribute a BD array in X-slice partitioning from the master to the X-workers.

**Syntax**

```
distrib\_x\_BD[1\|2\|3](Ny,Nz,Nyloc,Ns,aglob,aloc)
```

- **Ny**
  - ! global no. of grid points in y-dim
- **Nz**
  - ! global & local no. of grid points in z-dim
- **Nyloc**
  - ! local no. of grid points in y-dim
- **Ns**
  - ! local & global no. of chemical species dim
- **aglob(Ny,Nz,2,Ns)**
  - ! global BD array
- **alloc(Nyloc,Nz,2,Ns)**
  - ! local BD array on each X-worker

**Details**

These routines distribute a global BD array at the master into X-slice partitioning to the workers. A BD array is a 4D array in the dimensions x, z, 2, and number of chemical species or y, z, 2, and number of chemical species that contains data about each chemical species on the east and west or north and south boundaries of the model. For X-slices we will need the east and west boundaries for each slice. The following three versions are implemented:

- **v1** - distribution sending slice by slice using Ny messages
- **v2** - distribution sending one message to each X-worker combining the slices if more than one
- **v3** - using two calls to **MPI\_SCATTERV**

Note that the first routine is independent of the mapping while the second and third routines depend on the data types declared for a certain mapping in the function **CreateCommDatatypes** and therefore cannot be used with any other mapping unless the communication structures have been reimplemented.

**Location**

`mpi\_commlibrary.f`

Module: **XYCommunicationLibrary distrib\_y\_BD (page 30)**, **distrib\_x\_BD (page 42)**

---

**distrib\_y\_BD**

Distribute a BD array in Y-slice partitioning from the master to the Y-workers.

**Syntax**

```
distrib\_y\_BD[1\|2\|3](Nx,Nz,Nxloc,Ns,aglob,aloc)
```

- **Nx**
  - ! global no. of grid points in x-dim
- **Nz**
  - ! global & local no. of grid points in z-dim
- **Nxloc**
  - ! local no. of grid points in x-dim
- **Ns**
  - ! local & global no. of chemical species dim
- **aglob(Nx,Nz,2,Ns)**
  - ! global BD array
- **alloc(Nxloc,Nz,2,Ns)**
  - ! local BD array on each Y-worker
Details

These routines distribute a global BD array at the master into Y-slice partitioning to the workers. A BD array is a 4D array in the dimensions $x$, $z$, 2, and number of chemical species or $y$, $z$, 2, and number of chemical species that contains data about each chemical species on the east and west or north and south boundaries of the model. For Y-slices we will need the north and south boundaries for each slice. The following three versions are implemented:

v1 - distribution sending slice by slice using $N_x$ messages

v2 - distribution sending one message to each Y-worker combining the slices if more than one

v3 - using two calls to $\text{MPI}_{\text{SCATTERV}}$

Note that the first routine is independent of the mapping while the second and third routines depend on the data types declared for a certain mapping in the function $\text{CreateCommDatatypes}$ and therefore can not be used with any other mapping unless the communication structures have been reimplemented.

Location

$\text{mpi.commlibrary.f}$

Module: $\text{XYCommunicationLibrary distrib}_x\text{BD}$ (page 30), $\text{distrib}_y\text{hBD}$ (page 43)
3.3 HV-Partitioning

**init_hv**

Initialize and set up the communication for HV-partitioning.

**Syntax**

```
init_hv(Nx,Ny,Nz,Nzloc,Ncloc,Ns)
```

**Details**

This routine initializes the parallel communication for HV-partitioning by calling the three later described functions `init_hv_versions`, `CreateMap` and `CreateCommDataTypes`. Calling this routine instead of the others makes sure that the right modules are included.

**Location**

`mpi_communication.f`

Module: HVParallelCommunication

**See Also**

`init_hv_versions` (page 32), `CreateMap` (page 33), `CreateCommDataTypes` (page 34)

---

**init_hv_versions**

Initialize the communication versions to be used.

**Syntax**

```
init_hv_versions()
```

**Details**

This routine reads the file created by the setup program that specifies the fastest version numbers for the communication routines on the specific architecture using a set grid resolution and given number of processors. If the setup program has not been run the default version (version 1) will be used.

To decide on the versions during runtime either a few subroutines have to be provided that map the integers specified by the setup program (1, 2, 3) to the communication routine version. Another approach is hardcoding the best routines for the used architecture in the AQM-specific functions after testing the speed of each version using the setup programs.

**Location**
CreateMap

Create the mapping between H-slices and V-columns and processors as well as important global variables.

Syntax

CreateMap(Nx, Ny, Nz, Nzloc, Nloc)

INTEGER Nx        ! global no. of grid points in x-dim
INTEGER Ny        ! global no. of grid points in y-dim
INTEGER Nz        ! global no. of grid points in z-dim
INTEGER Nzloc     ! local no. of grid points in z-dim on H-workers
INTEGER Nloc      ! local no. V-columns on V-workers

Details

This routine is part of the module ParallelDataMap. It assigns the processors to be H-workers and
V-workers, respectively and keeps book about the assignments of the global H-slices and V-columns
to these workers. This includes the number of H-slices and V-columns assigned to each worker, their
global and local id, as well as the owner relationship to each H-slice and V-column.

Most global variables are set in this routine: the number of H and V-workers, the size of the local
arrays at the workers as well as mapping variables. The provided routine can serve as an example
on how to set up a different mapping. We have used the following global mapping variables:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner_of_hslicex(1..Nz)</td>
<td>specifies the process that gets the H-slice</td>
</tr>
<tr>
<td>owner_of_vcolx(1..Nx-Ny)</td>
<td>specifies the process that gets the V-column</td>
</tr>
<tr>
<td>no_of_hslicex(p)</td>
<td>number of H-slices owned by process p where p (\in) {1,..,NWorkers}</td>
</tr>
<tr>
<td>no_of_vcolx(p)</td>
<td>number of V-columns owned by process p where p (\in) {1,..,NWorkers}</td>
</tr>
<tr>
<td>local_hslicex(1..Nz)</td>
<td>local index of the global H-slice (index seen by the process which owns it)</td>
</tr>
<tr>
<td>local_vcolx(1..Nx-Ny)</td>
<td>local index of the global V-column (index seen by the process which owns it)</td>
</tr>
<tr>
<td>global_hslicex(p,1..Nzloc)</td>
<td>global index of a local H-slice on processor p</td>
</tr>
<tr>
<td>global_vcolx(p,1..Nloc)</td>
<td>global index of a local V-column on processor p</td>
</tr>
<tr>
<td>owned_hslicex(p,1..no_of_hslicex(p))</td>
<td>global H-slices owned by worker p</td>
</tr>
<tr>
<td>owned_vcolx(p,1..no_of_vcolx(p))</td>
<td>global V-columns owned by worker p</td>
</tr>
<tr>
<td>planar_vcolx(1..Nx-Ny,1..2)</td>
<td>global x and y index of V-column id</td>
</tr>
<tr>
<td>linear_vcolx(1..Nx,1..Ny)</td>
<td>global V-column id for column at given position in the global array</td>
</tr>
</tbody>
</table>

Location

mpi_util.f
Module: HVParallelDataMap
See Also

CreateCommDataTypes for XY-partitioning (page 21)

---

CreateCommDataTypes

Creates the data types used by the library routines for any communication.

Syntax

\[
\text{CreateCommDataTypes}(\text{Nx, Ny, Nz, Nzloc, Ncloc, Ns})
\]

INTEGER \( \text{Nx} \) ! global no. of grid points in x-dim
INTEGER \( \text{Ny} \) ! global no. of grid points in y-dim
INTEGER \( \text{Nz} \) ! global no. of grid points in z-dim
INTEGER \( \text{Nzloc} \) ! local no. of grid points in z-dim on H-workers
INTEGER \( \text{Ncloc} \) ! local no. of V-columns on V-workers
INTEGER \( \text{Ns} \) ! no. of chemical species to be monitored

Details

This routine creates the worker group used for communication during shuffling as well as the communication data types for each of the array types in the air quality model. There have to be slice-types and column-types to do the slice- and column-communication in distribution and gathering. Other data types created are the combined slices and columns for the single message distribution and gathering as well as the combined data to be sent during shuffling when using one message only. These data types represent parts of an array that can be located anywhere in the array specified by count numbers and displacements.

Most of these data types depend on the mapping function described above. The implementation can serve as an example on how to define these data types when using different mapping routines. Be aware that the combining of H-slices or V-columns to one data type for single message communication may not be possible if the mapping is irregular! In that case only the first version of communication routines can be used that distribute and gather the H-slices and V-columns slice by slice and column by column respectively.

Location

mpi_util.f
Module: HVCommDataTypes

See Also

CreateCommDataTypes for XY-partitioning (page 21)
3.4 HV Library

**distrib.h.4D**

Distribute a 4D array in H-slice partitioning from the master to the H-workers.

**Syntax**

\[
\text{distrib.h.4D}_v[1|2|3](\text{Nx, Ny, Nz, Nzloc, Ns, aglob, aloc})
\]

- `INTEGER Nx` ! global & local no. of grid points in x-dim
- `INTEGER Ny` ! global & local no. of grid points in y-dim
- `INTEGER Nz` ! global no. of grid points in z-dim
- `INTEGER Nzloc` ! local no. of grid points in z-dim
- `INTEGER Ns` ! local & global no. of chemical species dim
- `REAL aglob(Nx, Ny, Nz, Ns)` ! global 4D array
- `REAL aloc(Nx, Ny, Nzloc, Ns)` ! local 4D array on each H-worker

**Details**

These routines distribute a global 4D array at the master into H-slice partitioning to the H-workers. The following three versions are implemented:

- \(v1\) - distribution sending slice by slice using \(Nz\) messages
- \(v2\) - distribution sending one message to each H-worker combining the slices if more than one
- \(v3\) - using two calls to `MPI_SCATTERV`

Note that the first routine is independent of the mapping while the second and third routines depend on the data types declared for a certain mapping in the function `CreateCommDatatypes` and therefore cannot be used with any other mapping unless the communication structures have been reimplemented.

**Location**

`mpi.commlibrary.f`

Module: HVCommunicationLibrary

**See Also**

- `distrib.v.4D` (page 35), `distrib.x.4D` (page 22)

**distrib.v.4D**

Distribute a 4D array in V-column partitioning from the master to the V-workers.

**Syntax**

\[
\text{distrib.v.4D}_v[1|2|3](\text{Nx, Ny, Nz, Ncloc, Ns, aglob, aloc})
\]

- `INTEGER Nx` ! global no. of grid points in x-dim
- `INTEGER Ny` ! global no. of grid points in y-dim
- `INTEGER Nz` ! global & local no. of grid points in xy-dim
- `INTEGER Ncloc` ! local no. of grid points in v-dim
- `INTEGER Ns` ! local & global no. of chemical species dim
- `REAL aglob(Nx, Ny, Nz, Ns)` ! global 4D array
- `REAL aloc(1, Ncloc, Nz, Ns)` ! local 4D array on each V-worker
Details

These routines distribute a global 4D array at the master into V-column partitioning to the V-workers. The following three versions are implemented:

v1 - distribution sending column by column using Nx*Ny messages

v2 - distribution sending one message to each V-worker combining the columns if more than one

v3 - using two calls to MPI_SCATTERV

Note that the first routine is independent of the mapping while the second and third routines depend on the data types declared for a certain mapping in the function CreateCommDatatypes and therefore can not be used with any other mapping unless the communication structures have been reimplemented.

Location

mpi_commlibrary.f
Module: HWCommunicationLibrary

See Also

distrib_h_4D (page 35), distrib_v_4D (page 22)

gather_h_4D

Gather a 4D array at the master from H-slice partitioning at the H-workers.

Syntax

$\texttt{gather}_h\_4D\_v[1|2|3](\texttt{Nx},\texttt{Ny},\texttt{Nz},\texttt{Nzloc},\texttt{Ns},\texttt{aglob},\texttt{aloc})$

$\begin{align*}
\text{INTEGER } & \texttt{Nx} & \text{! global & local no. of grid points in x-dim} \\
\text{INTEGER } & \texttt{Ny} & \text{! global & local no. of grid points in y-dim} \\
\text{INTEGER } & \texttt{Nz} & \text{! global no. of grid points in z-dim} \\
\text{INTEGER } & \texttt{Nzloc} & \text{! local no. of grid points in z-dim} \\
\text{INTEGER } & \texttt{Ns} & \text{! local & global no. of chemical species dim} \\
\text{REAL } & \texttt{aglob(Nx,Ny,Nz,Ns)} & \text{! global 4D array} \\
\text{REAL } & \texttt{aloc(Nx,Ny,Nzloc,Ns)} & \text{! local 4D array on each H-worker} \\
\end{align*}$

Details

These routines gather the local 4D arrays at the H-workers into the global 4D array at the master. The following three versions are implemented:

v1 - master gathering, receiving slice by slice in Nz messages

v2 - master gathering, receiving one message from each H-worker combining the local slices if more than one

v3 - using two calls to MPI_GATHERV

Note that the first routine is independent of the mapping while the second and third routines depend on the data types declared for a certain mapping in the function CreateCommDatatypes and therefore can not be used with any other mapping unless the communication structures have been reimplemented.
Location

mpi_commlibrary.f
Module: HVCommunicationLibrary

See Also
gather_v4D (page 37), gather_x4D (page 23)

---

**gather_v4D**

Gather a 4D array at the master from V-column partitioning at the V-workers.

**Syntax**

```fortran
gather_v4D_v[1|2|3](Nx,Ny,Nz,Ncloc,Ns,aglob,aloc)
```

- INTEGER Nx ! global no. of grid points in x-dim
- INTEGER Ny ! global no. of grid points in y-dim
- INTEGER Nz ! global & local no. of grid points in z-dim
- INTEGER Ncloc ! local no. of grid points in xy-dim
- INTEGER Ns ! local & global no. of chemical species dim
- REAL aglob(Nx,Ny,Nz,Ns) ! global 4D array
- REAL aloc(1,Ncloc,Nz,Ns) ! local 4D array on each V-worker

**Details**

These routines gather the local 4D arrays at the V-workers into the global 4D array at the master. The following three versions are implemented:

- v1 - master gathering, receiving column by column in Nx Ny messages
- v2 - master gathering, receiving one message from each V-worker combining the local slices if more than one
- v3 - using two calls to MPI_GATHERV

Note that the first routine is independent of the mapping while the second and third routines depend on the data types declared for a certain mapping in the function CreateCommDatatypes and therefore can not be used with any other mapping unless the communication structures have been reimplemented.

---

**Location**

mpi_commlibrary.f
Module: HVCommunicationLibrary

See Also
gather_h4D (page 36), gather_v4D (page 24)

---

**shuffle_h2v4D**

Shuffle 4D arrays from h-partitioning on H-workers to v-partitioning on V-workers.

**Syntax**
shuffle\_v2h\_4D, v\{1|2|3\}(N_x, N_y, N_z, N\_loc, N\_cloc, N_s, aloc, aloc\_v)

\begin{verbatim}
INTEGER N_x ! global no. of grid points in x-dim
INTEGER N_y ! global no. of grid points in y-dim
INTEGER N_z ! global no. of grid points in z-dim
INTEGER N\_loc ! local no. of grid points in h-dim
INTEGER N\_cloc ! local no. of grid points in xy-dim
INTEGER N_s ! local & global no. of chemical species dim
REAL aloc(N_x, N_y, N\_loc, N_s) ! local 4D array on each H-worker
REAL aloc\_v(1, N\_cloc, N_z, N_s) ! local 4D array on each V-worker
\end{verbatim}

Details

These routines rearrange the local 4D in h-partitioning at the H-workers to v-partitioning at the V-workers. The following three versions are implemented:

\begin{itemize}
  \item v1 - distribution using copying into a buffer and sending blocks, one message from each H-worker to each V-worker
  \item v2 - distribution sending one message from each H-worker to each V-worker combining the necessary data
  \item v3 - using copying and MPI\_ALLTOALL
\end{itemize}

Note that the first and third routines are independent of the mapping while the second routine depends on the data types declared for a certain mapping in the function CreateCommDatatypes and therefore can not be used with any other mapping unless the communication structures have been reimplemented.

Location

\texttt{mpi\_commlibrary.f}

Module: HVCommunicationLibrary

See Also

shuffle\_v2h\_4D (page 38), shuffle\_x2y\_4D (page 24)

---

**shuffle\_v2h\_4D**

Shuffle 4D arrays from v-partitioning on V-workers to h-partitioning on H-workers.

Syntax

\begin{verbatim}
shuffle\_v2h\_4D, v\{1|2|3\}(N_x, N_y, N_z, N\_loc, N\_cloc, N_s, aloc, aloc\_v)
\end{verbatim}

\begin{verbatim}
INTEGER N_x ! global no. of grid points in x-dim
INTEGER N_y ! global no. of grid points in y-dim
INTEGER N_z ! global no. of grid points in z-dim
INTEGER N\_loc ! local no. of grid points in h-dim
INTEGER N\_cloc ! local no. of grid points in xy-dim
INTEGER N_s ! local & global no. of chemical species dim
REAL aloc(N_x, N_y, N\_loc, N_s) ! local 4D array on each H-worker
REAL aloc\_v(1, N\_cloc, N_z, N_s) ! local 4D array on each V-worker
\end{verbatim}

Details
These routines rearrange the local 4D in v-partitioning at the V-workers to h-partitioning at the H-workers. The following three versions are implemented:

v1 - distribution using copying into a buffer and sending blocks, one message from each V-worker to each H-worker

v2 - distribution sending one message from each V-worker to each H-worker combining the necessary data

v3 - using copying and MPI_ALLTOALL

Note that the first and third routines are independent of the mapping while the second routine depends on the data types declared for a certain mapping in the function CreateCommDatatypes and therefore can not be used with any other mapping unless the communication structures have been reimplemented.

Location

mpi_commlibrary.f
Module: HVCommunicationLibrary

See Also

shuffle_h2v_4D (page 37), shuffle_v2x_4D (page 25)

distrib_h_2D

Distribute a 2D array in H-slice partitioning from the master to the H-workers.

Syntax

distrib_h_2D(v1|2|3) (Nx,Ny,aglob)

INTEGER Nx       ! global & local no. of grid points in x-dim
INTEGER Ny       ! global & local no. of grid points in y-dim
REAL aglob(Nx,Ny) ! global & local 2D array

Details

This routine distributes a global 2D array at the master into H-slice partitioning to the H-workers using broadcast. Currently there are no 2D arrays that have to be distributed in H-slice form, but we provided an implementation.

Location

mpi_commlibrary.f
Module: HVCommunicationLibrary

See Also

distrib_v_2D (page 39), distrib_x_2D (page 26)

distrib_v_2D

Distribute a 2D array in V-column partitioning from the master to the V-workers.
Syntax

\[
distrib.y.2D.v[1|2|3](Nx,Ny,Ncloc,aglob,alloc)
\]

\[
\begin{align*}
\text{INTEGER } Nx & \quad \text{! global no. of grid points in x-dim} \\
\text{INTEGER } Ny & \quad \text{! global no. of grid points in y-dim} \\
\text{INTEGER } Ncloc & \quad \text{! local no. of grid points in v-dim} \\
\text{REAL } aglob(Nx,Ny) & \quad \text{! global 2D array} \\
\text{REAL } aloc(1,Ncloc) & \quad \text{! local 2D array on each V-worker}
\end{align*}
\]

Details

These routines distribute a global 2D array at the master into V-column partitioning to the V-workers. The following three versions are implemented:

\[
\begin{align*}
v1 & \quad \text{- distribution sending column by column using Nx-Ny messages} \\
v2 & \quad \text{- distribution sending one message to each V-worker combining the columns if more than one} \\
v3 & \quad \text{- using two calls to } \text{MPI}_{-}SCATTERV
\end{align*}
\]

Note that the first routine is independent of the mapping while the second and third routines depend on the data types declared for a certain mapping in the function \text{CreateCommDatatypes} and therefore can not be used with any other mapping unless the communication structures have been reimplemented.

Location

\text{mpi\_commlibrary.f}  
Module: \text{HVCommunicationLibrary}

See Also

distrib.h.2D (page 39), distrib.y.2D (page 26)

---

\[
distrib.y.2DN
\]

Distribute a 2DN array in V-column partitioning from the master to the V-workers.

Syntax

\[
distrib.y.2DN.v[1|2|3](Nx,Ny,Ncloc,Ns,aglob,alloc)
\]

\[
\begin{align*}
\text{INTEGER } Nx & \quad \text{! global no. of grid points in x-dim} \\
\text{INTEGER } Ny & \quad \text{! global no. of grid points in y-dim} \\
\text{INTEGER } Ncloc & \quad \text{! local no. of grid points in v-dim} \\
\text{INTEGER } Ns & \quad \text{! local & global no. of chemical species dim} \\
\text{REAL } aglob(Nx,Ny,Ns) & \quad \text{! global 2DN array} \\
\text{REAL } aloc(1,Ncloc,Ns) & \quad \text{! local 2DN array on each V-worker}
\end{align*}
\]

Details

These routines distribute a global 2DN array at the master into V-column partitioning to the V-workers. A 2DN array is a 3D array in the dimensions x, y, and number of chemical species. The following three versions are implemented:
v1 - distribution sending column by column using Nx-Ny messages

v2 - distribution sending one message to each V-worker combining the columns if more than one

v3 - using two calls to MPI_SCATTERV

Note that the first routine is independent of the mapping while the second and third routines depend on the data types declared for a certain mapping in the function CreateCommDatatypes and therefore cannot be used with any other mapping unless the communication structures have been reimplemented.

Location

mpi_commmlibrary.f
Module: HVCommunicationLibrary

See Also
distrib_y2DN (page 28)

distrib_h_3D

Distribute a 4D array in H-slice partitioning from the master to the H-workers.

Syntax

distrib_h_4D_y[1|2|3](Nx,Ny,Nz,Nzloc,aglob,aloc)

INTEGER Nx ! global & local no. of grid points in x-dim
INTEGER Ny ! global & local no. of grid points in y-dim
INTEGER Nz ! global no. of grid points in z-dim
INTEGER Nzloc ! local no. of grid points in z-dim
REAL aglob(Nx,Ny,Nz) ! global 3D array
REAL aloc(Nx,Ny,Nzloc) ! local 3D array on each H-worker

Details

These routines distribute a global 3D array at the master into H-slice partitioning to the H-workers. The following three versions are implemented:

v1 - distribution sending slice by slice using Nz messages

v2 - distribution sending one message to each H-worker combining the slices if more than one

v3 - using two calls to MPI_SCATTERV

Note that the first routine is independent of the mapping while the second and third routines depend on the data types declared for a certain mapping in the function CreateCommDatatypes and therefore cannot be used with any other mapping unless the communication structures have been reimplemented.

Location

mpi_commmlibrary.f
Module: HVCommunicationLibrary
See Also
distrib_v.3D (page 42), distrib_x.3D (page 28)

---

**distrib_v.3D**

Distribute a 3D array in V-column partitioning from the master to the V-workers.

**Syntax**

```fortran
   distrib_v.3D, v[1|2|3] (Nx,Ny,Nz,Nloc,aglob,alloc)
```

- INTEGER Nx   ! global no. of grid points in x-dim
- INTEGER Ny   ! global no. of grid points in y-dim
- INTEGER Nz   ! global & local no. of grid points in xy-dim
- INTEGER Nloc ! local no. of grid points in v-dim
- REAL aglob(Nx,Ny,Nz) ! global 3D array
- REAL alloc(1,Nloc,Nz) ! local 3D array on each V-worker

**Details**

These routines distribute a global 3D array at the master into V-column partitioning to the V-workers. The following three versions are implemented:

- v1 - distribution sending column by column using Nx*Ny messages
- v2 - distribution sending one message to each V-worker combining the columns if more than one
- v3 - using two calls to MPI_SCATTERV

Note that the first routine is independent of the mapping while the second and third routines depend on the data types declared for a certain mapping in the function **CreateCommDatatypes** and therefore cannot be used with any other mapping unless the communication structures have been reimplemented.

**Location**

`mpi_commlibrary.f`

Module: HVCommunicationLibrary

See Also
distrib_h.3D (page 41), distrib_v.3D (page 29)

---

**distrib_xh_BD**

Distribute a BDx array in H-slice partitioning from the master to the H-workers.

**Syntax**

```fortran
   distrib_xh_BD, v[1|2|3] (Nx,Nz,Nloc,Ns,aglob,alloc)
```

42
INTEGER Nx  ! global & local no. of grid points in x-dim  
INTEGER Nz  ! global no. of grid points in z-dim  
INTEGER Nzloc ! local no. of grid points in z-dim  
INTEGERNs  ! local & global no. of chemical species dim  
REAL aglob(Nx,Nz,2,Ns)  ! global BDx array  
REAL aloc(Nx,Nzloc,2,Ns)  ! local BDx array on each H-worker

Details  
These routines distribute a global BDx array at the master into H-slice partitioning to the H-workers. A BDx array is a 4D array in the dimensions x, z, 2, and number of chemical species that contains data about each chemical species on the east and west boundaries of the model. The following three versions are implemented:

  v1 - distribution sending slice by slice using Nz messages  
  v2 - distribution sending one message to each H-worker combining the slices if more than one  
  v3 - using two calls to MPI_SCATTERV

Note that the first routine is independent of the mapping while the second and third routines depend on the data types declared for a certain mapping in the function CreateCommDatatypes and therefore can not be used with any other mapping unless the communication structures have been reimplemented.

Location

mpi_commLibrary.f  
Module: HVCommunicationLibrary

See Also

distrib_yh_BD (page 43), distrib_x_BD (page 30)

distrib_yh_BD

Distribute a BDy array in H-slice partitioning from the master to the H-workers.

Syntax

distrib_yh_BD,v[1|2|3](Ny,Nz,Nzloc,Ns,aglob,aloc)

  INTEGER Ny  ! global & local no. of grid points in y-dim  
  INTEGER Nz  ! global no. of grid points in z-dim  
  INTEGER Nzloc ! local no. of grid points in z-dim  
  INTEGERNs  ! local & global no. of chemical species dim  
  REAL aglob(Ny,Nz,2,Ns)  ! global BDy array  
  REAL aloc(Ny,Nzloc,2,Ns)  ! local BDy array on each H-worker

Details

These routines distribute a global BDy array at the master into H-slice partitioning to the H-workers. A BDy array is a 4D array in the dimensions y, z, 2, and number of chemical species that contains data about each chemical species on the north and south boundaries of the model. The following three versions are implemented:

43
v1 - distribution sending slice by slice using Nz messages
v2 - distribution sending one message to each H-worker combining the slices if more than one
v3 - using two calls to MPI_SCATTERV

Note that the first routine is independent of the mapping while the second and third routines depend on the data types declared for a certain mapping in the function CreateCommDatatypes and therefore can not be used with any other mapping unless the communication structures have been reimplemented.

Location
mpi_commlibrary.f
Module: HVCommunicationLibrary

See Also
distrib_xh_BD (page 42), distrib_y_BD (page 30)