RALEF-2D code documentation

M.M. Basko
*KIAM, Moscow and GSI, Darmstadt**
(Dated: June 29, 2014)

Contents

*Electronic address: `mmbasko@gmail.com`; URL: `http://www.basko.net`

1. HOW TO RUN THE RALEF CODE

1.1. File names and directory structure

Assume that the principal project directory for simulating a particular problem, from which the RALEF executable is to be executed, has a name `*/p/`. Then, in the standard configuration the structure of this directory should be

```
*/p/code/
*/p/input/
*/p/in2d
```
(1.1)

Here

- the subdirectory `*/p/code/` contains all the FORTRAN source-code files `'*.f'`,
- the subdirectory `*/p/input/` contains all the files `'*.ii'` with the input information that might be needed during the code run (like the GLT-table files `'table-glt-eos.ii'` and `'table-glt-tcrad.ii'`, the file `'xspmonbc.ii'` with a tabular form of the incident x-ray spectrum, etc.),

- and ‘in2d’ is the principal input file where the values of various parameters from the namelist/input/ are assigned.

Here and below, to distinguish the alone-standing file names from other types of variables, we surround them by single quote marks like ‘ ’.

Note that file ‘in2d’ is the only input file which has no extension, and which must be placed into the principal project directory */p/. All the other files with the numeric-data input information must have the extension .ii and be placed into the subdirectory input/, which must (!) be created because it appears explicitly in the corresponding FORTRAN statements. And only if no input-information files ‘*.ii’ are required for the current job, the subdirectory input/ may be omitted. In contrast, the subdirectory code/ does not appear explicitly in the FORTRAN statements — hence its presence is never obligatory.

Once the RALEF executable is executed from the project directory */p/ (by, say, the UNIX command

```
nohup ./code/a.out > aa.dat &
```

when code/ is the compilation directory), all the output files will be written into the same project directory */p/, and will all have either the extension .dat or the extension .vtk.

1.2. Compilation and execution

RALEF-2D is a parallelized code, with a hybrid MPI/OpenMP parallelization scheme. Hence, it can be run in two different modes: an MPI mode, and a non-MPI mode. In addition, any of these two modes can be combined with the OpenMP compiler option. Thus, the code in the non-MPI mode compiled without OpenMP will be executed sequentially; the code in the non-MPI mode compiled with the OpenMP option will be executed in the OpenMP-parallelized mode.

For the compilation and execution of the RALEF code, go through the following steps.

1. *In the MPI mode:*

(a) uncomment all the MPI-declarations in the FORTRAN files ‘f01_main_ccMPI.f’ and ‘f09_rad_ccMPI.f’ by deleting all combinations of the 4 symbols !MPI at the start of every code line where this combination is encountered; change the file names to standard ones ‘f01_main.f’ and ‘f09_rad.f’;

(b) gather the 13 FORTRAN source-code files

```
f00_comod.f
f01_main.f        (MPI-declarations are present)
f02_init.f
f03_bound.f
f04_hydro.f
f05_remap.f
f06_eos.f
f07_lnktbls.f
f08_util.f
f09_rad.f         (MPI-declarations are present)
f10_taskinpt.f
f11_taskdepo.f
f12_taskout.f
```

into the source-code subdirectory code/.

In the non-MPI mode: gather the 13 FORTRAN source-code files

```
f00_comod.f
f01_main_ccMPI.f     (MPI-declarations are commented out)
f02_init.f
f03_bound.f
f04_hydro.f
f05_remap.f
f06_eos.f
f07_lnktbls.f
f08_util.f
f09_rad_ccMPI.f      (MPI-declarations are commented out)
f10_taskinpt.f
f11_taskdepo.f
f12_taskout.f
```

into the source-code subdirectory `code/`,

or

gather the 14 FORTRAN source-code files

```
f00_comod.f
f01_main.f       (MPI-declarations are present)
f02_init.f
f03_bound.f
f04_hydro.f
f05_remap.f
f06_eos.f
f07_lnktbls.f
f08_util.f
f09_rad.f        (MPI-declarations are present)
f10_taskinpt.f
f11_taskdepo.f
f12_taskout.f
f13_mpi-dummy.f
```

into the source-code subdirectory `code/`.

2. If no tabular EOS or opacities are used, replace the file '`f07_lnktbls.f`' with the dummy file '`f07_lnktbls-dummy.f`'.

3. By using an appropriate compiler,

(a) compile the three files '`f00_comod.f`', '`f08_util.f`' and '`f13_mpi-dummy.f`' (if needed), which contain various modules;

(b) compile and link the rest of the code.

UNIX EXAMPLES:

```
ifort -c -openmp f00_comod.f f08_util.f f13_mpi-dummy.f
ifort -openmp *.f


mpif90 -c -openmp f00_comod.f f08_util.f
mpif90 -openmp *.f
```

4. Gather all the needed numeric-data files with the input information for the RALEF code (like ‘table-glt-eos.ii’, ‘table-glt-tcrad.ii’, ‘xspmonbc.ii’, etc.) into the subdirectory */p/input/.

5. Run the code by executing the executable from the project directory */p/, which contains the principal input file ‘in2d’.

   UNIX EXAMPLE with 12 MPI-tasks:

```
mpirun -np 12 ./code/a.out > aa.dat &
```

1.3. Assigning the principal problem parameters in the input file ‘in2d’

The values of the principal run-control parameters are assembled into the namelist/input/ and are assigned by editing the text file ‘in2d’. The first line of this file before the operator $INPUT contains the job title. An example of the ‘in2d’ file for simulation of an empty cylindrical hohlraum is given below.

```
Au cyl.hohlraum:  case 01:  empty, 1 nu-group
&input !  Begin NAMELIST "input"
nrestart=0
!  Units of measurement (in terms of CGS units):
unilngth=1.d-1
unitime=1.d-8
unimass=1.d-3
unitemp=1.60217733d-9

!  Geometry and mesh:
igeom =3 !  r-theta polar mesh with iradial=0
iradial=0
nblks=1
nprts(1,1)=3 !  3 parts along theta
ncell(1,1,1)=54,24,180
ncell(1,2,1)=40
xxl(1,1,1)=24.d0, 78.d0, 102.d0, 336.d0 !  degrees
xxl(1,2,1)=0.35d0, .365d0 !  radial dimensions

!  Material properties:
matnum(1,1)=1,1,1
proprty(1,1)=7.d0 !  EOS type
proprty(2,1)=.1d0 !  rho00, used for wt(i) in REZONE
proprty(4,1)=1.2d0 !  strong shock parameter
proprty(5,1)=1.d0 !  m-number in GLT tables
proprty(27,1)=196.97d0, 79.d0
proprty(30,1)=7.d0, 1.d0 !  conduction
proprty(37,1)=-.5d0
proprty(40,1)=7.d0, 1.d0 !  opacity
proprty(46,1)=3.d0 !  laser absorption

!  Initial and boundary conditions:
rho0(1,1)=19.d0, 1.d-4, 19.d0
```

```
pr0(1,1)=1.d-7, 1.d-7, 1.d-7
pbc(1,1,1)=1.d-7, 1.d-7, 1.d-7
pbc(1,2,1)=1.d-7, 1.d-7, 1.d-7
pbc(1,3,1)=1.d-7
pbc(1,4,1)=1.d-7

!  Radiation, thermal treatment:
itemp=4
nfreqsets=2
nfreqs=1,200
kradSn=3
ifshadow=1
iflasdep=1
iradfbc=0
iradbbc=0
TEMPFLR=3.d-5
TEMPSNS=2.d-3
EPS0SSI=.2d0
EPS1SSI=.1d0

!  Runtime, printout control:
twfin=.2d0
ncycmax=1
ntty=100
twmovi=0.0d0, 1.d7, .05d0
twprnt=0.d0, 1.d35, .05d0
twfilm=0.d0, 1.d1, 1.3d2, 2.d35
twspctr=0.d0, .1d9, .05d9
twdump=1.d2, 1.d35, 1.d2

!  ALE, rezone control:
iorder=1
alecoef=.99d0
dt0=1.d-6
dtmin=1.d-60
dtmax=1.d-4
dtfac=0.5

nadvskp=1
wtampl=6.d0
nsmooth=4
itrezn=5

ibcrezn(1,1)=2,1,1,1
crezsm=.4d0
prezsm=1.5d0
arezsm=.03d0
nrezbay=12
arezbay=.6d0
crezbay=.4d0
```

```
/ !  End NAMELIST "input"
```

The meaning of all the parameters that can be set via this namelist is explained in the documentation file ‘00glossar_RALEF.txt’.

1.4. Mesh construction

Typically, the user is supposed to select one of the preprogrammed mesh types from the RALEF mesh library by assigning the corresponding value of the `igeom` parameter in the namelist `input` (file ‘in2d’). All possible values of the `igeom` parameter are listed in section 9 below, together with a detailed description of the corresponding mesh and its other control parameters that are to be specified in the namelist `input`.

If the user does not find the needed type of mesh in the current version of the RALEF mesh library, he has to assign a new value `igeom` = `xxxx` and write the corresponding mesh construction routines `subroutine MSHPxxxx` and `subroutine MSHBxxxx(XV)` by analogy with those already available in file ‘f02_init.f’ — thus enriching the RALEF mesh library.

1.5. Equation of state and opacities

1. Analytical equation of state and/or opacities

If one of the preprogrammed analytical models is used for the equation of state, or the thermal conduction coefficient, or the radiation absorption coefficient, then the parameters of the corresponding analytical model are set up in the input file ‘in2d’. These parameters are all contained in the principal material-property array `proprty(1:100,1:30)`; the meaning of all relevant elements of this array is explained in section 12 below.

2. GLT-tabular equation of state and/or opacities

If either for the equation of state, or for the thermal conduction coefficient, or for the radiation absorption coefficients a tabular option in the form of the general logarithmic tables (GLT) is chosen by setting `proprty(1,imat)` = 7.0 [or `proprty(30,imat)` = 7.0, or `proprty(40,imat)` = 7.0], then the files ‘table-glt-eos.ii’ and/or ‘table-glt-tcrad.ii’ with corresponding tables must be generated. This is accomplished by running separate code packages GLT-EOS and GLT-TCR: the GLT-EOS package generates the file ‘table-glt-eos.ii’ , which contains the GLT database for tabular EOS; the GLT-TCR package generates the file ‘table-glt-tcrad.ii’, which contains the GLT database for tabular thermal conduction coefficient and/or tabular Rosseland, Planckian and spectral opacities. In this way tabular opacities and/or thermal conductivity can be combined with any analytical EOS and vice versa.

2. HOW TO GENERATE THE GLT TABLES

3. HOW TO SET UP LASER DRIVE

3.1. General steps

To activate the laser energy deposition, go through the following steps:

1. in the `namelist/input/`:
   (a) assign `iflasdep=1` to activate the non-refractive laser deposition model # 1, or assign `iflasdep=2` to activate the short-characteristics refractive (RSC) laser deposition model # 2
   (b) set the logical flag `ifshdwlas=.false.` if the search for shadows cast by protruding parts of external boundaries is not needed;
   (c) assign `proprty(46,imat)` = 3 to use the analytical inverse-bremsstrahlung absorption coefficient for the laser light;
2. set all the parameters that specify the laser beam configuration [number of beams `nblas`, propagation direction `(omeblasx(iblas),omeblasy(iblas))` of every beam `iblas`, etc.] by editing Step 1 in the `subroutine LASINPT`, file '`f10_taskinpt.f`';
3. select a preprogrammed option [by assigning the corresponding value to the parameter `itprcaslas(iblas)` at Step 1 in the `subroutine LASINPT`] — or program a new one — for the normalized temporal power profile of every laser beam by editing the `function FLASTPRO` in file "`f10_taskinpt.f`";
4. select a preprogrammed option [by assigning the corresponding value to the parameter `isprcaslas(iblas)` at Step 1 in the `subroutine LASINPT`] — or program a new one — for the normalized spatial power profile of every laser beam by setting a corresponding value of the `iwwprof` parameter in `function FLASSPRO` in file "`f10_taskinpt.f`";

3.2. How to use laser power profiles provided in numerical form

To use a temporal laser-power profile, provided in a numerical form, go through the following steps:

1. prepare the file '`input/lastpro.ii`' with the corresponding numeric input data;
2. assign a positive non-zero value to the variable `ntproflas` in the subroutine `LASINPT`, file '`f10_taskinpt.f`', which must be equal to the total number of different numeric t-profiles to be used in the current job;
3. distribute the `ntproflas` numeric profiles among the `nblas` laser beams by reprogramming the user-defined section at the beginning of subroutine `FLASTPRO`, file '`f10_taskinpt.f`', where the variable `itprocase` must be set equal to 9, and the variable `itproseqnum` must be assigned the sequential number of the numeric t-profile from file '`input/lastpro.ii`' to be applied to the current laser beam `iblas`.

The external input data file '`input/lastpro.ii`' must have the following structure. Every of the `ntproflas` profiles must be specified as two columns of numbers (the column of times t and the column of powers p_t), recognized by the `G` format descriptor and preceded by a header of comment lines. The header may consist of an arbitrary number of record lines beginning with the sign `#`. No empty lines are allowed in the header. The first numeric profile may have no header. The two-column numeric sections of file '`input/lastpro.ii`' are allowed to be interspersed by empty record lines.

To synchronize the time units in the '`input/lastpro.ii`' file and the RALEF code, the first header (before the first numeric profile) should (but must not) contain a comment line of the form

`#time_unit` u_t

where u_t is a number (recognized by the `G` format descriptor) giving the time unit (in seconds) used in the first numeric column of the file '`input/lastpro.ii`'. If u_t is not specified, the currently used time unit of the RALEF code is assumed.

Example of an '`input/lastpro.ii`' file with two t-profiles where time is measured in nanoseconds:

```
# Example of file with two t-profiles
#time_unit 1.d-9
#t-profile #1:
0.d0 0.d0
1.e-1 1.e0
1.e0 1.e0

# t-profile #2:
1.2e-12 1.e-12
1.e0 5.e0

2.e0 3.e0
3.e0 3.e0
```

The time values in the first numeric column of file '`input/lastpro.ii`' must be monotonically increasing. If the first time t_1 is negative, the whole time sequence is shifted forward by $-t_1$, so that all times become non-negative. Finally, all the times are rescaled to the currently used RALEF time unit.

The power values p_t may be given in arbitrary units. Having been read out, they are always normalized to the peak value of $p_{t,max} = 1$. Thus normalized powers are integrated in time (already in the RALEF units) to provide the "energy" value

$$e = \int p_t \, dt$$

for each numeric t-profile `i`, which is stored as the value of variable `elastpro(i)`.

3.3. How to emulate a cylindrical laser beam with a conical one in the (r, z)-geometry

In the (r, z)-geometry cylindrical laser beams propagate strictly along the rotation axis Z, i.e. have $\Omega_{las,R} = 0$. As a matter of convenience, for such beams we use the flux density F_{00} itself (which has the dimensionality of [erg cm^{-2} s^{-1}]) to specify the incident laser intensity in subroutine `LASINPT` rather than the modified flux $\tilde{F}_{las} = RF_{las}$ with the dimensionality of [erg cm^{-1} s^{-1}]. The radial dependence of the laser beam intensity is prescribed by a dimensionless spatial-profile function $p_r(s)$, where s is the distance to laser beam axis.

In some cases it may be desirable to tilt such a cylindrical beam by a physically negligible angle (say, by an angle of the order of $\lesssim 10^{-8}$) with respect to the rotation axis — i.e. to emulate a physically cylindrical laser beam with a mathematically conical one. Indeed, this can be done by setting

$$\Omega_{las,Z} = \pm 1, \qquad \Omega_{las,R} = -10^{-8}, -10^{-9}, \ldots \tag{3.1}$$

(so that $|\Omega_{las,Z}^2 + \Omega_{las,R}^2 - 1| < 10^{-16}$) and by simultaneously modifying the radial profile $p_r(s)$ as

$$p_r(s) \to s p_r(s). \tag{3.2}$$

Then the original value of the incident flux F_{00} [erg cm^{-2} s^{-1}] may be left unchanged.

4. HOW TO SET UP VARIOUS NUMERICAL DIAGNOSTICS AND DATA OUTPUT

4.1. Angular distribution of emission in selected spectral band

The selected spectral band

$$\texttt{k1nusel} \le \texttt{kfreq} \le \texttt{k2nusel} \tag{4.1}$$

is set up by specifying (in file 'in2d') the values of parameters k1nusel and k2nusel from the range $1 \le$ k1nusel $\le$ k2nusel $\le$ nfreqs(1) for the main frequency set # 1; here kfreq is the index for frequency groups from this set.

To enact the printout of angular distribution of the thermal radiation, coming out of the simulated region in this spectral band, one has

1. to assign some positive value to the integer parameter nnuseltet ≥ 1 at Step 7 of subroutine TASKINPT in file 'f10_taskinpt.f', and

2. to set up the direction (Omnustetx,Omnustety) of the reference axis, with respect to which the diagnostic (polar) angle θ should be measured, at the same Step 7 of subroutine TASKINPT.

The angular distribution of the emitted radiation, integrated over the full physical boundary of the simulated region, is written into the file 'enuseloutdir.dat' at times, specified by the twfilm array.

Once it is greater than zero, the specific value of nnuseltet, assigned by the user, is not relevant: it is always adjusted to the current order kradSn of the S_n quadrature,

$$\texttt{nnuseltet} = \begin{cases} 4 \times \texttt{kradSn}, & \texttt{iradial} = 0, \\ 2 \times \texttt{kradSn}, & \texttt{iradial} = 1, 2. \end{cases} \tag{4.2}$$

The value of nnuseltet will be automatically changed at code restart whenever the value of kradSn is changed. In such a case accumulation of the emitted energy in the array enuseltet(:) is interrupted, and the subsequent dump files 'dp2dxxxx' become incompatible with the previous ones.

The user-provided components (Omnustetx,Omnustety) of the vector along the reference axis must not necessarily be normalized: the normalization is automatically done by the code. For iradial $= 1, 2$ the reference axis must be colinear with the rotation axis; in this case the emitted radiation is collected from the entire 4π of the solid angle, i.e. with participation of all the S_n directions. For iradial $= 0$ the reference axis must be colinear either with the x-axis or with the y-axis; in this case the emitted radiation is collected only from the equatorial (with respect to the z-axis) band of the S_n directions.

5. GRID NOTATION

5.1. General quantities

The entire computational mesh is assumed to be comprised of nblks ≥ 1 individual *blocks*. Each block iblk=1,2, ... has a topologically rectangular mesh with n1(iblk) *physical cells* along mesh direction 1, and n2(iblk) physical cells along mesh direction 2; see Fig. 5.1. Clearly, each block iblk has (n1(iblk) + 1 *physical vertices* along mesh direction 1, and n2(iblk) + 1 physical vertices along mesh direction 2. To these are added 4 belts of *ghost*

vertices and *ghost cells* along the 4 edges of each rectangular block. Together with the ghost cells and vertices, a block `iblk` requires a memory of

$$\texttt{msz(iblk)} = [\texttt{n1(iblk)} + 3][\texttt{n2(iblk)} + 3] \tag{5.1}$$

locations for any scalar quantity.

The global index for a scalar quantity in block `iblk` is

$$\texttt{I} = j \cdot [\texttt{n1(iblk)} + 3] + i + 1 + \texttt{mob(iblk)}, \tag{5.2}$$

where $i = 0, 1, \ldots, \texttt{n1(iblk)} + 2$ is the local index along mesh direction 1, and $j = 0, 1, \ldots, \texttt{n2(iblk)} + 2$ is the local index along mesh direction 2 for a cell (i, j) in block `iblk`,

$$\texttt{mob(iblk)} = \sum_{k=1}^{\texttt{iblk}-1} \texttt{msz}(k). \tag{5.3}$$

The global indices for the first and second components of a two-component vector quantity are

$$\texttt{I}_x = j \cdot [\texttt{n1(iblk)} + 3] + i + 1 + 2 \cdot \texttt{mob(iblk)} = \texttt{I} + \texttt{mob(iblk)}, \tag{5.4}$$

$$\texttt{I}_y = \texttt{I}_x + \texttt{msz(iblk)}. \tag{5.5}$$

If the global index I for a scalar quantity is known, then the local indices i and j can be restored as

$$j = [\texttt{I-mob(iblk)} - 1]/[\texttt{n1(iblk)} + 3], \tag{5.6}$$

$$i = \texttt{I-mob(iblk)} - 1 - j \cdot [\texttt{n1(iblk)} + 3]. \tag{5.7}$$

The total number `msoccp` of memory locations required by a scalar quantity is given by

$$\texttt{msoccp} = \texttt{mob(nblks)+msz(nblks)}, \tag{5.8}$$

where `nblks` is the total number of blocks.

In block-local subroutines, a block-local single index

$$\texttt{i} = j \cdot [\texttt{n1(iblk)} + 3] + i + 1 = \texttt{I} - \texttt{mob(iblk)} \tag{5.9}$$

is used to order scalar quantities. Analogously, the block-local indices for two components of a vector field are

$$\texttt{i}_x = j \cdot [\texttt{n1(iblk)} + 3] + i + 1 = \texttt{i}, \tag{5.10}$$

$$\texttt{i}_y = \texttt{i}_x + \texttt{msz(iblk)}. \tag{5.11}$$

Each block has 4 edges `ib=1,2,3,4` and 4 corners `ic=1,2,3,4`. The numbering convention for the block edges `ib` is shown in Fig. 5.1: if index i runs from left to right, and index j runs from bottom to top, then `ib=1` is the bottom edge, `ib=2` is the top edge, `ib=3` is the left edge, and `ib=4` is the right edge of the block. The numbering convention for the block corners `ic` is as follows: `ic=ib`, where `ib` is the number of the bottom edge after the block is rotated such that the corner `ic` becomes a lower-left corner of the block; see Fig. 5.2.

Notice that, while we have a single row of ghost vertices along each edge of a block, we have one row of ghost cells along edges `ib` = 1 and 3, and two rows of ghost cells along edges `ib` = 2 and 4; see Fig. 5.1. Hence, in certain cases we have to distinguish between the *ghost cells of the 1-st row* and the *ghost cells of the 2-nd row*.

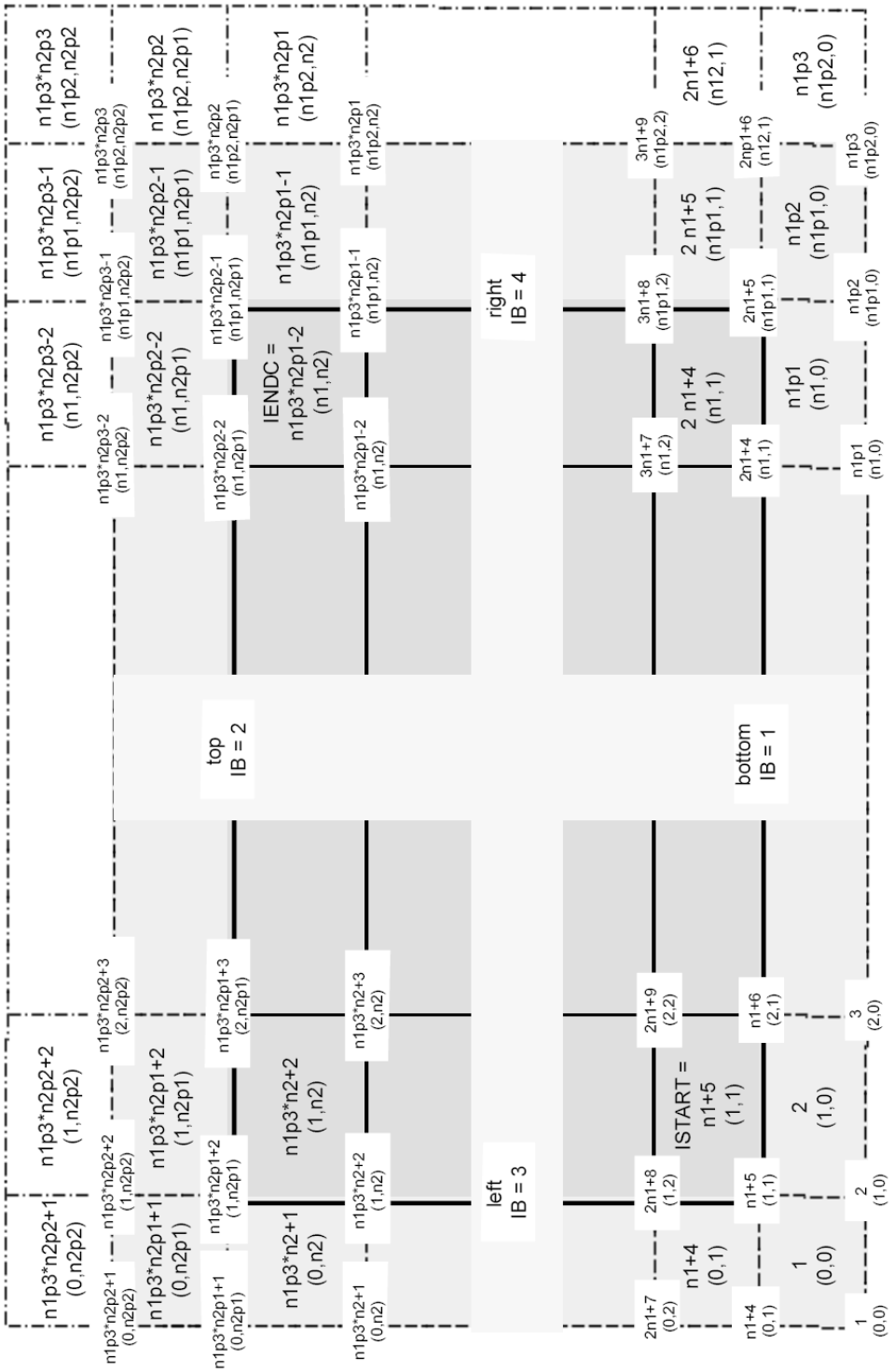


FIG. 5.1: Mesh indices and parameters inside a single block.

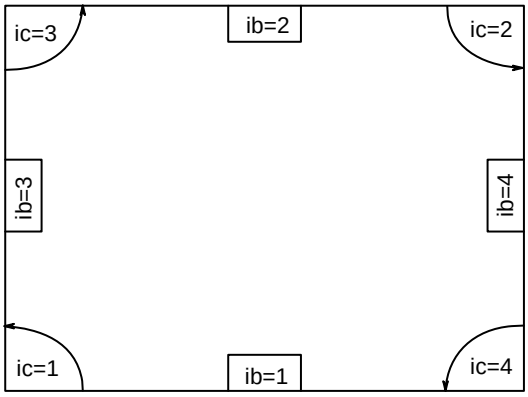


FIG. 5.2: Edge and corner numbering in a block.

5.2. Offsets for neighbors of a vertex and/or a cell

5.3. Offsets for do-loops along block edges

To perform `do`-loops along block edges, arrays `i1bc(ib,iblk)`, `i2bc(ib,iblk)`, and `i3bc(ib,iblk)` are defined as

```
i1bc(1,iblk)=1,
i1bc(2,iblk)=[n1(iblk)+3][n2(iblk)+1]+1,
i1bc(3,iblk)=1,
i1bc(4,iblk)=n1(iblk)+2,
```
(5.12)

```
i2bc(1,iblk)=n1(iblk)+3,
i2bc(2,iblk)=[n1(iblk)+3][n2(iblk)+2],
i2bc(3,iblk)=[n1(iblk)+3][n2(iblk)+2]+1,
i2bc(4,iblk)=msz(iblk)-1,
```
(5.13)

```
i3bc(1,iblk)=1,
i3bc(2,iblk)=1,
i3bc(3,iblk)=n1(iblk)+3,
i3bc(4,iblk)=n1(iblk)+3.
```
(5.14)

Here `i1bc(ib,iblk)` is the block-local single index of the *first* ghost cell along edge `ib` of block `iblk`; `i2bc(ib,iblk)` is the block-local single index of the *last* ghost cell along edge `ib` of block `iblk`; `i3bc(ib,iblk)` is the single-index *stride* along edge `ib` of block `iblk`; see Fig. 5.3. Then, a `do`-loop

```
do i=i1bc(ib,iblk),i2bc(ib,iblk),i3bc(ib,iblk)
...
enddo
```

will sweep over all ghost cells along edge `ib` of block `iblk`.

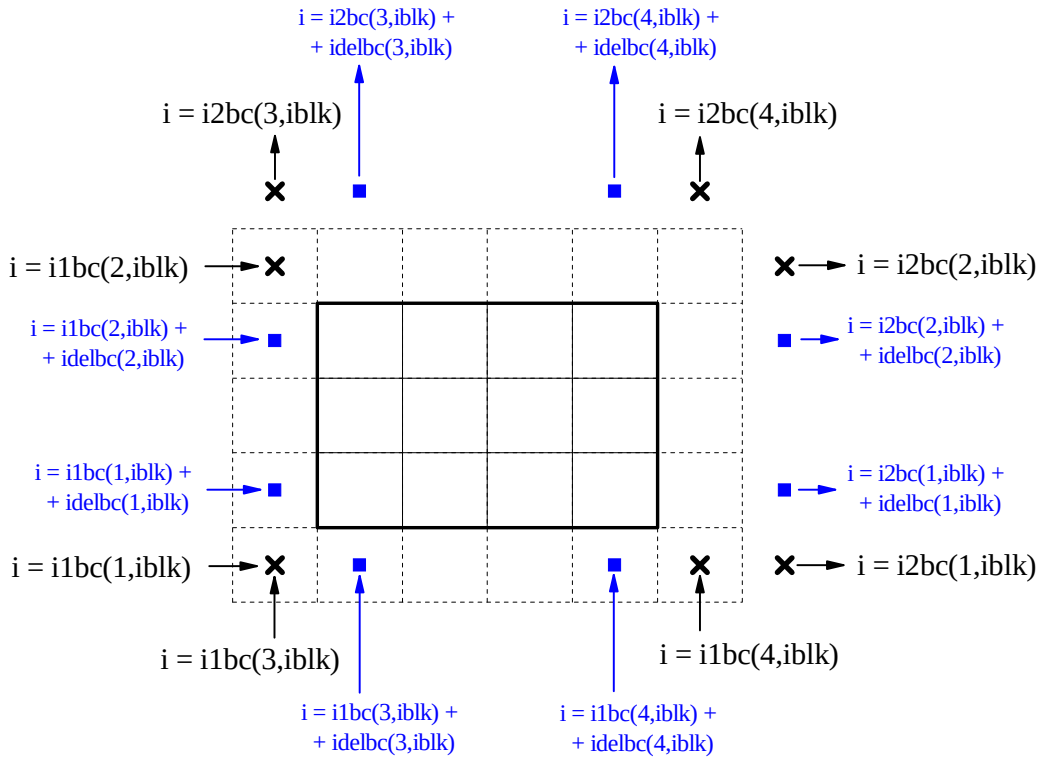


FIG. 5.3: Offsets for `do`-loops along block edges.

To perform `do`-loops over physical cells along block edges, an array

```
idelbc(1,iblk)=n1(iblk)+3,
idelbc(2,iblk)=-n1(iblk)-3,
idelbc(3,iblk)=1,
idelbc(4,iblk)=-1.
```
(5.15)

of offsets towards physical domain is defined; see Fig. 5.3. Thus, a `do`-loop

```
i1=i1bc(ib,iblk)+i3bc(ib,iblk)+idelbc(ib,iblk)
i2=i2bc(ib,iblk)-2*i3bc(ib,iblk)+idelbc(ib,iblk)

do i=i1,i2,i3bc(ib,iblk)
...
enddo
```

sweeps over all physical cells along edge `ib` of block `iblk`.

`Do`-loops over edge vertices and boundary cell faces can be arranged by using the same arrays. For example, a double `do`-loop over all boundary cell faces (or physical vertices) of block `iblk` may be programmed as

```
do ib=1,4

   i1=i1bc(ib,iblk)+i3bc(ib,iblk)+mod(ib,2)*idelbc(ib,iblk)
   i2=i2bc(ib,iblk)-2*i3bc(ib,iblk)+mod(ib,2)*idelbc(ib,iblk)
```

```
      do i=i1,i2,i3bc(ib,iblk)
      ...
      enddo

  enddo
```

6. INTERBLOCK COMMUNICATION

6.1. General quantities

To provide control over edge orientation, the following 4-value arrays are permanently defined as

$$\texttt{ia(ib)} = \begin{pmatrix} 1 \\ 1 \\ 2 \\ 2 \end{pmatrix}, \quad \texttt{iend(ib)} = \begin{pmatrix} -1 \\ 1 \\ 1 \\ -1 \end{pmatrix}, \quad \texttt{ileft(ib)} = \begin{pmatrix} 3 \\ 4 \\ 2 \\ 1 \end{pmatrix}, \quad \texttt{irght(ib)} = \begin{pmatrix} 4 \\ 3 \\ 1 \\ 2 \end{pmatrix}. \tag{6.1}$$

The value of `ia(ib)` gives mesh direction along edge `ib`; `ileft(ib)` gives the neighboring edge number on the left side of edge `ib` (oriented such that the block body lies above edge `ib`); `irght(ib)` gives the neighboring edge number on the right side of edge `ib`; `iend(ib)` gives the direction to the primary (left) corner of edge `ib`. Arrays `ileft(ib)` and `irght(ib)` have also zeroth components `ileft(0)=1` and `irght(0)=1`.

If a block `iblk` locks onto block `jblk` along its edge `ib`, which physically coincides with the edge `jb` of block `jblk`, then we have

$$\texttt{jblk= nbc(1,ib,iblk)}, \qquad \texttt{jb = nbc(2,ib,iblk)} \tag{6.2}$$

$$\texttt{nedg(m,iblk)} = \begin{cases} \texttt{n1(iblk)} + 3 \equiv n1p3, & m = 1, \\ \texttt{n2(iblk)} + 3 \equiv n2p3, & m = 2, \end{cases} \tag{6.3}$$

$$\texttt{nedgf(m,iblk) = nedg(m,iblk)} - 2. \tag{6.4}$$

Block communication is accomplished by using predefined strides and offsets for the global index

$$\texttt{I} = \texttt{mob(iblk)} + j \cdot [\texttt{n1(iblk)} + 3] + i + 1, \quad \begin{cases} i = 0, 1, \ldots, n1p2, \\ j = 0, 1, \ldots, n2p2. \end{cases} \tag{6.5}$$

The acceptor block `IBLK` receives information for its ghost cells along edge `IB` from the donor block `JBLK` along its matching edge `JB`. Stride along edge `IB` of the acceptor block `IBLK` is given by

$$ms1(ib, iblk) = i3bc(ib, iblk) = \begin{pmatrix} 1 \\ 1 \\ n1p3 \\ n1p3 \end{pmatrix}. \tag{6.6}$$

Stride along edge `JB` of the donor block `JBLK` is given by

$$ls1(ib, iblk) = io \cdot i3bc(jb, jblk), \tag{6.7}$$

where `IO = +1` for mesh-parallel block contact (like edge `IB=1` to edge `JB=2`), and `IO =` -1 for mesh-anti-parallel block contact (like edge `IB=1` to edge `JB=1`).

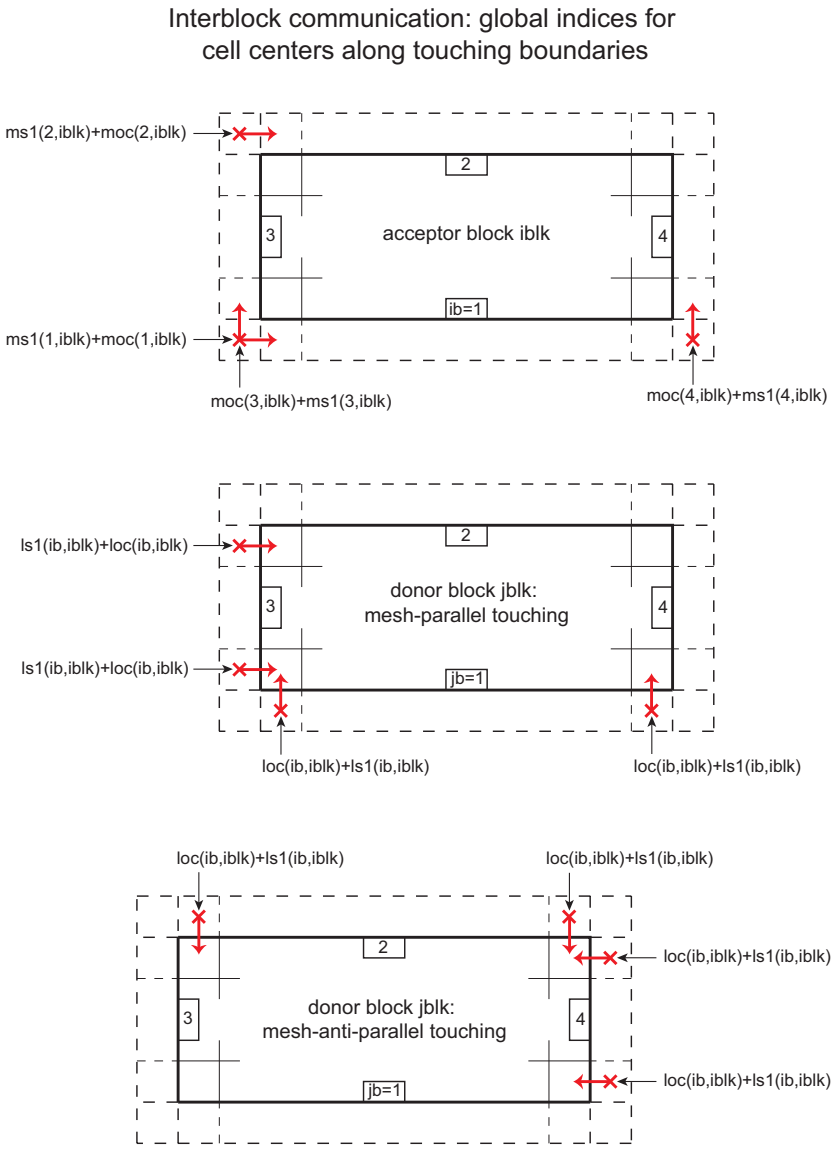


FIG. 6.1: Global indices for cell-centered quantities along communicating block boundaries.

Global offsets for the acceptor block `IBLK` are given by

$$moc(ib, iblk) = mob(iblk) + \begin{pmatrix} 0 \\ n2p1 \cdot n1p3 \\ -n1p2 \\ -1 \end{pmatrix}, \tag{6.8}$$

$$mov(ib, iblk) = mob(iblk) + \begin{pmatrix} 0 \\ n2p2 \cdot n1p3 \\ -n1p2 \\ 0 \end{pmatrix}, \tag{6.9}$$

$$moe(ib,iblk) = mob(iblk) + \begin{pmatrix} n1p3 \\ n2p1 \cdot n1p3 \\ -n1p1 \\ -1 \end{pmatrix}, \tag{6.10}$$

$$mof(ib,iblk) = mob(iblk) + \begin{pmatrix} 1 + msz(iblk) \\ 1 + n2p1 \cdot n1p3 + msz(iblk) \\ 1 \\ n1p2 \end{pmatrix}. \tag{6.11}$$

`MOC(IB,IBLK)+MS1(IB,IBLK)` is the first cell center to accept information along edge `IB` (see Fig. 6.1). Analogously, `MOV(IB,IBLK)+MS1(IB,IBLK)` is the first vertex to accept information along edge `IB` (see Fig. 6.2), `MOE(IB,IBLK)+MS1(IB,IBLK)` is the first vertex to accept information along nearest sequence parallel to edge `IB` (see Fig. 6.3), `MOF(IB,IBLK)+MS1(IB,IBLK)+ MOB(IBLK)` is the first face, perpendicular to edge `IB`, which is to accept information along edge `IB` (see Fig. **??**). If one is interested in physical vertices only, then `MOE(IB,IBLK)+2*MS1(IB,IBLK)` is the first physical vertex along edge `IB` of block `IBLK`; respectively, `MOE(IB,IBLK)+[NEDG(IA(IB)),IBLK)-1]*MS1(IB,IBLK)` is the last physical vertex along edge `IB` of block `IBLK`. All the above m-quantities are defined for all boundaries of all blocks, irrespective of whether they are, or are not, in contact with other blocks.

Strides and global offsets for donor blocks with mesh-parallel contact are

$$ls1(ib,iblk) = i3bc(jb,jblk) = \begin{cases} 1, & jb = 1, \\ 1, & jb = 2, \\ n1(jblk) + 3, & jb = 3, \\ n1(jblk) + 3, & jb = 4, \end{cases} \tag{6.12}$$

$$loc(ib,iblk) = mob(jblk) + \begin{cases} n1p3, & jb = 1, \\ n2 \cdot n1p3, & jb = 2, \\ -n1p1, & jb = 3, \\ -2, & jb = 4, \end{cases} \tag{6.13}$$

$$lov(ib,iblk) = mob(jblk) + \begin{cases} 2 \cdot n1p3, & jb = 1, \\ n2 \cdot n1p3, & jb = 2, \\ -n1, & jb = 3, \\ -2, & jb = 4, \end{cases} \tag{6.14}$$

$$loe(ib,iblk) = mob(jblk) + \begin{cases} n1p3, & jb = 1, \\ n2p1 \cdot n1p3, & jb = 2, \\ -n1p1, & jb = 3, \\ -1, & jb = 4, \end{cases} \tag{6.15}$$

For l-quantities the values $n1p3 = n1(jblk) + 3$, $n2p2 = n2(jblk) + 2$ and similar are calculated for the donor block `JBLK`; the columns of 4 values are ordered after donor edges `JB =1,2,3,4`.

Strides and global offsets for donor blocks with mesh-anti-parallel contact are given by

$$ls1(ib,iblk) = -i3bc(jb,jblk) = \begin{cases} -1, & jb = 1, \\ -1, & jb = 2, \\ -n1p3, & jb = 3, \\ -n1p3, & jb = 4., \end{cases} \tag{6.16}$$

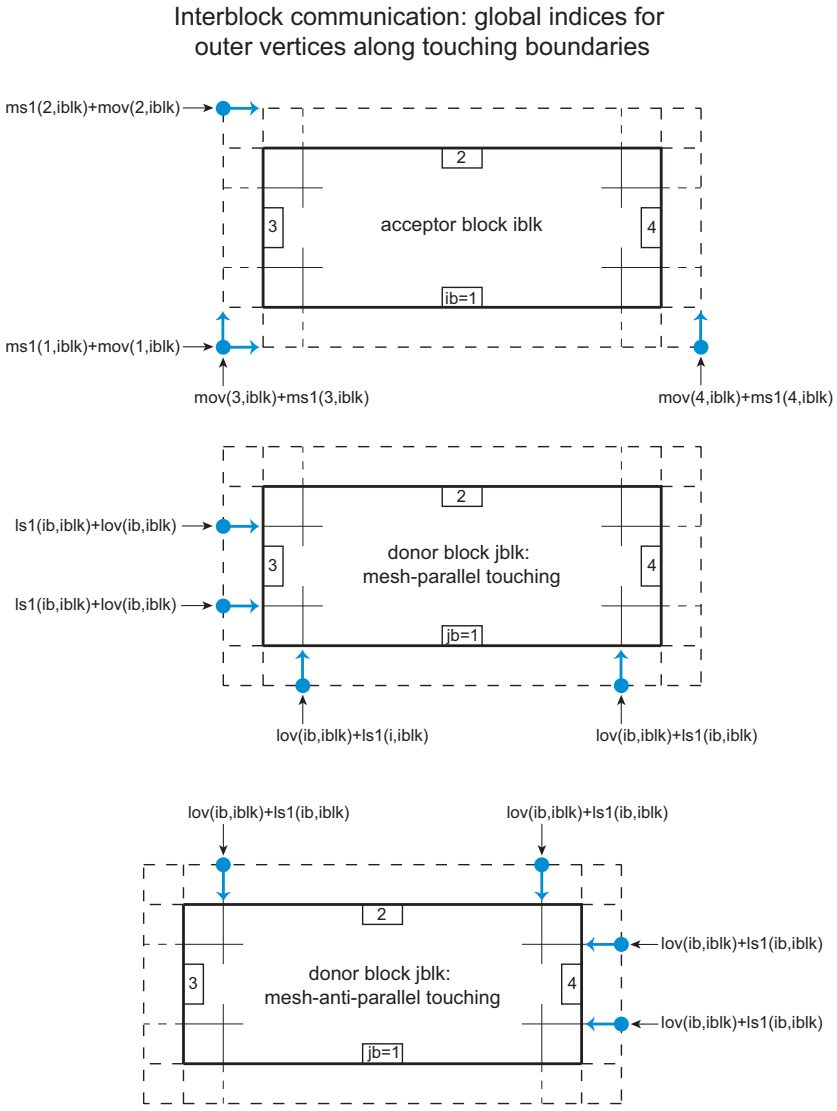


FIG. 6.2: Global indices for outer (along ghost edges) vertices along communicating block boundaries.

$$loc(ib, iblk) = mob(jblk) + \begin{cases} 2 \cdot n1p3, & jb = 1, \\ n2p1 \cdot n1p3, & jb = 2, \\ n2p2 \cdot n1p3 + 2, & jb = 3, \\ n2p3 \cdot n1p3 - 2, & jb = 4, \end{cases} \tag{6.17}$$

$$lov(ib, iblk) = mob(jblk) + \begin{cases} 3 \cdot n1p3 + 1, & jb = 1, \\ n2p1 \cdot n1p3 + 1, & jb = 2, \\ n2p3 \cdot n1p3 + 3, & jb = 3, \\ n2p4 \cdot n1p3 - 2, & jb = 4, \end{cases} \tag{6.18}$$

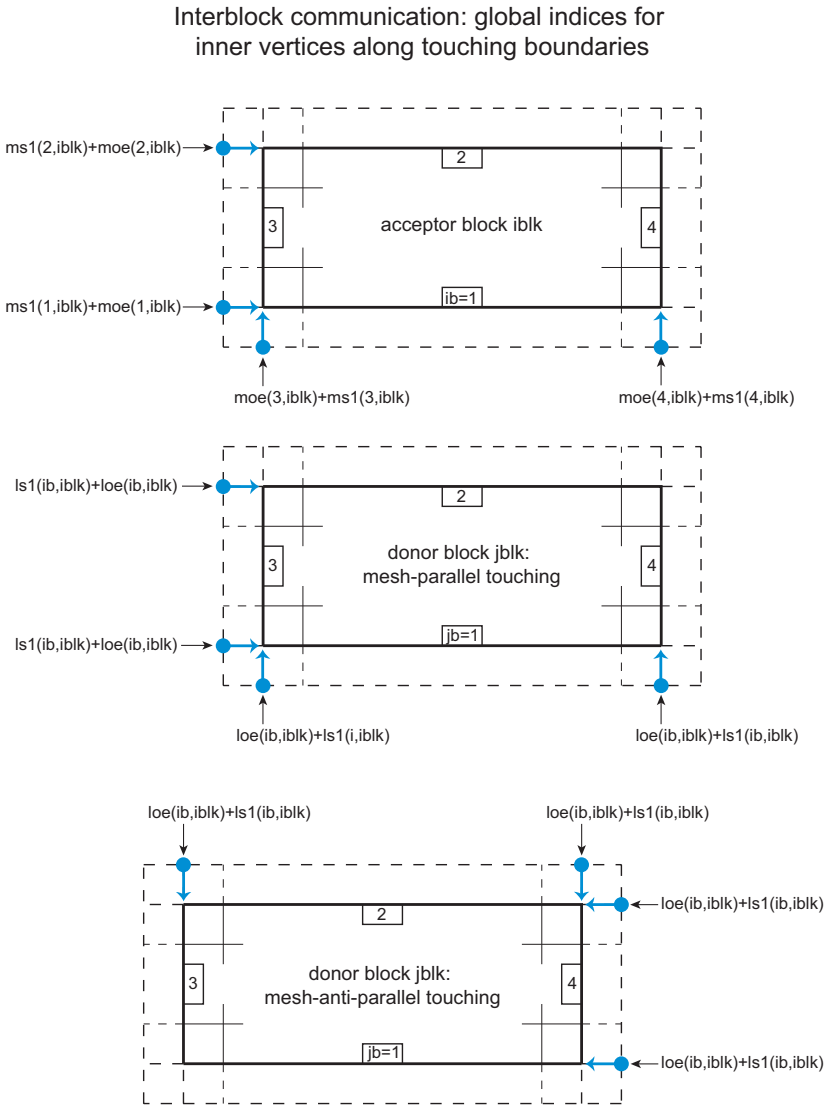


FIG. 6.3: Global indices for inner (along physical edges) vertices along communicating block boundaries.

$$loe(ib, iblk) = mob(jblk) + \begin{cases} 2 \cdot n1p3 + 1, & jb = 1, \\ n2p2 \cdot n1p3 + 1, & jb = 2, \\ n2p3 \cdot n1p3 + 2, & jb = 3, \\ n2p4 \cdot n1p3 - 1, & jb = 4, \end{cases} \tag{6.19}$$

6.2. 4-block meeting point

If a corner `IB` in a block `IBLK` is a 4-block meeting point (a 4-bk point, no void is allowed!), the flag `I4BK(IB,IBLK)` is set equal to 1. In this case the vertex coordinates and physical parameters of all ghost cells around the 4-bk point are uniquely defined by the corresponding physical cells in corresponding donor blocks.

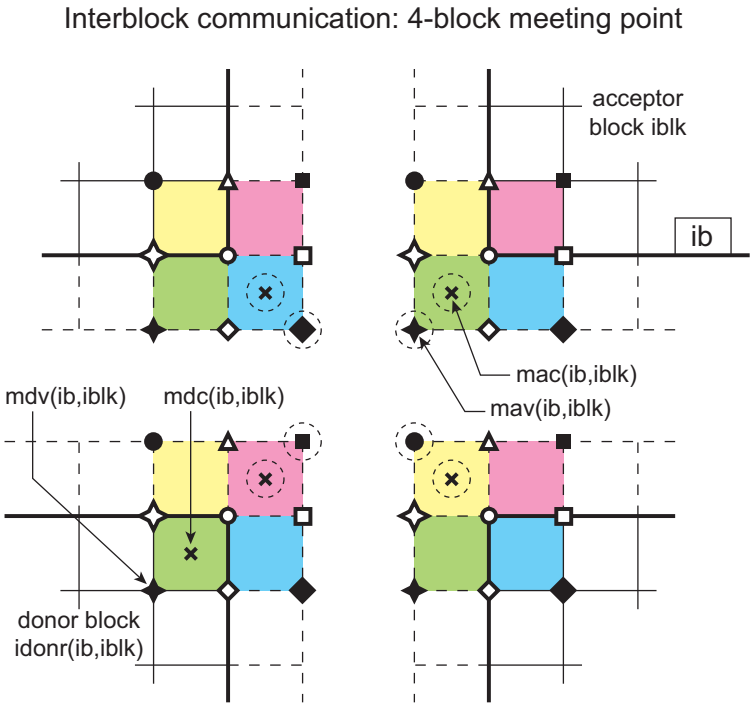


FIG. 6.4: Block communication at a 4-block meeting point. Physically identical vertices are marked with the same symbols. Vertices and cell centers, which require special action, are dash-encircled.

Regular boundary-to-boundary communication procedure: correct values are assigned to all acceptor ghost cell parameters except for (i) the ghost-corner vertex `MAV(IB,IBLK)`, and (ii) the ghost-corner cell center `MAC(IB,IBLK)` (see Fig. 6.4) — which get some spurious values.

Special action: (i) the x, y coordinates of the ghost-corner `MAV(IB,IBLK)` in the acceptor block `IBLK` are set equal to the x, y coordinates of the physical vertex `MDV(IB,IBLK)` in the donor block `IDONR(IB,IBLK)`; (ii) the cell-centered values of ghost-corner cell `MAC(IB,IBLK)` in the acceptor block `IBLK` are set equal to the cell-centered values in the physical cell `MDC(IB,IBLK)` of the donor block `IDONR(IB,IBLK)`; see Fig. 6.4.

As a result, the physical corner vertex at a 4-bk point "sees" identical patterns around itself from any of the 4 contacting blocks.

6.3. 3-block meeting point

If a corner `IB` in a block `IBLK` is a 3-block meeting point with no void left(a 3-bk point), the flag `I3BK(IB,IBLK)` is set equal to 1 (this is done for contacting corners in each of the 3 contacting blocks). In this case the vertex coordinates and physical parameters of all ghost cells around the 3-bk point — with the exception of ghost-corner cells — are also uniquely defined by the corresponding physical cells in corresponding donor blocks. The ghost-corner cell of each contacting block degenerates into a single cell edge (i.e. has zero volume), along which the physical-corner cells of two other blocks contact one another.

Regular boundary-to-boundary communication procedure: correct values are assigned to all acceptor ghost cell parameters except for (i) the ghost-corner vertex `MAV(IB,IBLK)`, and (ii) the ghost-corner cell center `MAC(IB,IBLK)` (see Fig. 6.5) — which get some spurious

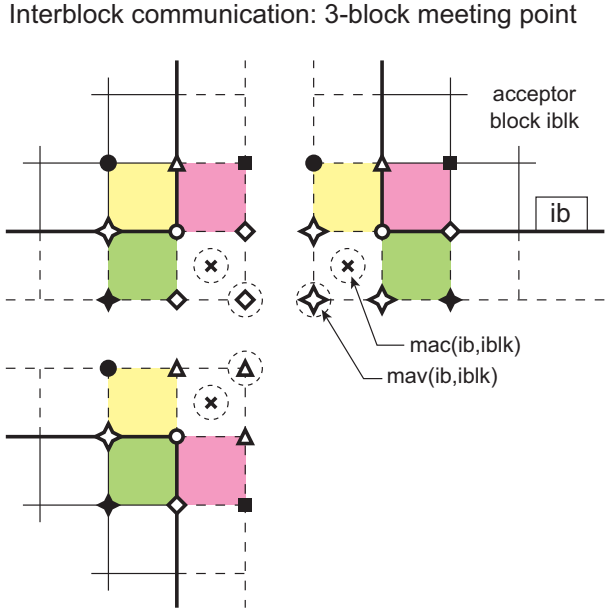


FIG. 6.5: Block communication at a 3-block meeting point. Physically identical vertices are marked with the same symbols. Vertices and cell centers, which require special action, are dash-encircled.

values.

Special action: (i) the x, y coordinates of the ghost-corner `MAV(IB,IBLK)` are set identical to those of one of the two neighboring ghost vertices (which coincide in the physical space!), and (ii) the cell-centered values in the ghost-corner cell `MAC(IB,IBLK)` are determined by interpolation between the cell-centered values of the two neighboring cells.

As a result, in the 4-surrounding-cells picture, the physical 3-bk corner vertex sees different patterns around itself from the 3 different contacting blocks. It sees identical patterns only when the degenerate ghost-corner cell is ignored in each of the 3 contacting blocks.

6.4. 3-block-void meeting point

When 3 blocks meet in an L-shape configuration and a void is left (a 3-vd meeting point), one block has a central position and a "donor-donor" contact; the other two meeting blocks, which have a "void-donor" type of contact, are diagonally opposite to one another (similar to the 4-bk meeting point); see Fig. 6.6. In such a case the flag `I3VD(IC,IBLK)` is set equal to 1 in each of the two "void-donor" blocks (but not in the central "donor-donor" block!) for a corresponding corner `IC` at the 3-vd meeting point. To mark the corresponding 3-vd corner `IC` in the central "donor-donor" block `IBLK`, another special flag `I3DD(IC,IBLK)` is introduced (in the CAVEAT-TR version) and set equal to 1 for that corner.

Regular boundary-to-boundary communication procedure: correct values are assigned to the acceptor ghost cell parameters except for (i) the ghost-corner vertex `MAV(IB,IBLK)`, and (ii) the ghost-corner cell center `MAC(IB,IBLK)` (see Fig. 6.6) — which may (or may not, depends on the order of assignment sequence) get some spurious values.

Special action is taken only for the ghost-corner cells of the 2 "void-donor" blocks:

1. the x, y coordinates of the ghost vertex `mav(ib,iblk)` in the acceptor block `iblk` are

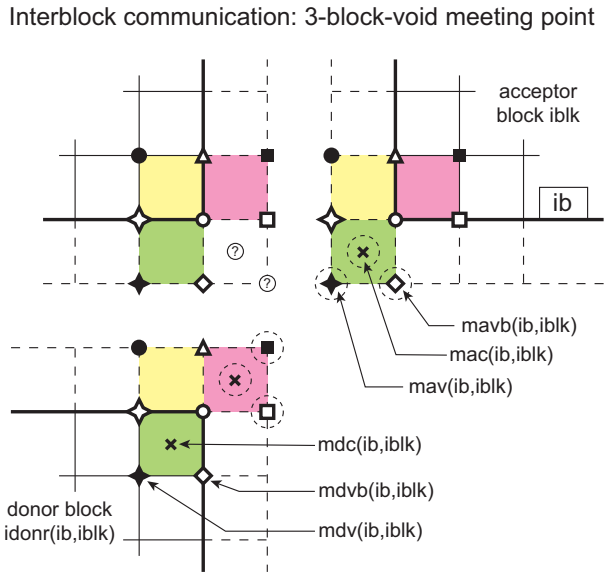


FIG. 6.6: Block communication at a 3-block-void meeting point. Physically identical vertices are marked with the same symbols. Vertices and cell centers, which require special action, are dash-encircled.

set equal to the x, y coordinates of the physical vertex `mdv(ib,iblk)` in the donor block `idonr(ib,iblk)`;

2. the cell-centered values of the ghost-corner cell `mac(ib,iblk)` in the acceptor block `iblk` are set equal to the cell-centered values in the physical cell `mdc(ib,iblk)` of the donor block `idonr(ib,iblk)`;

3. in addition, the x, y coordinates of the ghost vertex `mavb(ib,iblk)` in the acceptor block `iblk` are set equal to the x, y coordinates of the physical vertex `mdvb(ib,iblk)` in the donor block `idonr(ib,iblk)` (see fig. 6.6); to ensure the correct sign of the ghost-face normal velocity u^* (at the face connecting the corner `ib` with the ghost vertex `mavb(ib,iblk)`), the value of `nsig3(ib,iblk)` is set equal either to $+1$ (the unit normals `fn(i,m)` of communicating faces have the same direction in space), or to -1 (the corresponding unit normals have opposite directions in space).

As a result, in the 4-surrounding-cells picture, the physical 3-vd corner vertex sees different patterns around itself from the 3 different contacting blocks. It sees identical patterns only when the ghost-corner cell is ignored in the central "donor-donor" block, and corresponding corner-neighboring ghost cells are ignored in each of the 2 "void-donor" blocks.

Important: a mesh with a 3-vd meeting point on a reflective boundary is not allowed! Such a mesh would lead to two contradictory prescriptions for calculating the coordinates of the ghost vertex `MAVB(IB,IBLK)`.

Important: since no special action is taken for the central "donor-donor" block, the coordinates of its `MAV(IB,IBLK)` ghost vertex, and the cell-centered quantities in the `MAC(IB,IBLK)` ghost cell retain their spurious values!

6.5. Changes in RALEF-2D

1. An additional flag `i3dd(ib,iblk)` is introduced, which is set equal to 1 for the contacting corner `ib` of the "donor-donor" central block `iblk` in a 3-block-void-meeting situation.

2. The values of `mav(ib,iblk)` and `mac(ib,iblk)` are calculated for all blocks in the problem — similar to `ms1(ib,iblk)`, `mov(ib,iblk)`, `moc(ib,iblk)`, ...

7. FLAG ARRAY `IFLG(I)`

7.1. Flags and masks

Certain key information on individual cells and vertices is contained in a compact form in a mesh-wide integer-4 array `iflg(I)`. For every vertex (cell) `I`, the integer value of `iflg(I)` is composed of a set of *flags*; each flag occupies only several bits of `iflg(I)`. The value of a flag `xxx` can be retrieved by applying the logical `iand` operator to `iflg(I)` with a corresponding mask `mskxxx`

$$\texttt{xxx} = \texttt{iand(iflg(I),mskxxx)}. \tag{7.1}$$

Evidently, any single-bit flag can have a value of either 0 or 2^{k-1}, where k is the sequential number of the corresponding bit. Table I lists the flags and their masks used in the RALEF code.

In more detail, the rules for setting the key flags in the `iflg(I)` array are as follows:

- flag `mat`, mask `mskmat`:
  - flag `mat` is cell-centered, significant in both physical and ghost cells;
  - is set =`matnum` in all physical cells;
  - is set =`matnum` in all non-corner ghost cells along the boundaries with `ibc=3` (outflow) and `ibc=4` (inflow);
  - is set =`matnum` of the corresponding donor cell in all ghost cells (corner + non-corner) of the 1st row along the interblock boundaries with `ibc=5`; for the ghost corner cell with `i3bk=1` the donor is the neighboring physical corner cell of the same block;
- flag `gst`, mask `mskgst`:
  - flag `gst` is cell-centered, serves to distinguish between physical and ghost cells; significant in both physical and ghost cells;
  - is set =1 in all ghost cells (both 1st and 2nd rows);
- flag `bnd`, mask `mskbnd`:
  - flag `bnd` is cell-centered, significant in ghost cells only;
  - is set =`ib` in all ghost cells (1st + 2nd rows) along block edge `ib`; in the corner cells it takes the values of 3 or 4;
- flag `fix`, mask `mskfix`:

TABLE I: Masks for reading the `iflg(I)` array.

mask	bits	value	description; allocation
`mskmat`	1-6	$2^6 - 1$	Yields material number; cell-centered.
`mskgst`	7	2^6	Is 'on' in ghost cells; cell-centered.
`msklvx`	8	2^7	Indicates a strictly Lagrangian vertex that cannot be rezoned; vertex-centered. Note that `msklvx` and `mskivx` are never both 'on'.
`msklf1`	9	2^8	Indicates a no-fluxing cell face along mesh direction 1; face-centered.
`msklf2`	10	2^9	Indicates a no-fluxing cell face along mesh direction 2; face-centered.
`mskivx`	11	2^{10}	Indicates a vertex on a Lagrangian (material) interface that can be tangentially rezoned; vertex-centered.
`msktr1`	12	2^{11}	Indicates that a vertex can be tangentially rezoned along mesh direction 1; vertex-centered.
`msktr2`	13	2^{12}	Indicates that a vertex can be tangentially rezoned along mesh direction 2; vertex-centered.
`mskfix`	14	2^{13}	Indicates that a vertex remains fixed in space; vertex-centered.
`mskrev`	15	2^{14}	Mask for Sesame EOS
`mskheb`	16	2^{15}	Mask for rezone inhibition on a burn front.
`mskmx`	17	2^{16}	Indicates that a cell has a mixed EOS; cell-centered.
`mskbnd`	18-20	7×2^{17}	Yields edge number of a 1st-row ghost cell; cell-centered.
`mskgcc`	21	2^{20}	Indicates that a cell is a corner ghost cell; cell-centered.
`mskbov`	22	2^{21}	Indicates that a vertex is a physical vertex along a block boundary; vertex-centered.
`mskpag`	23	2^{22}	Indicates that a cell is a Physically Associated Ghost Cell (PAGC); cell-centered.
`mskcnv`	24	2^{23}	Indicates that a vertex is a physical block-corner vertex; vertex-centered.

  - flag `fix` is vertex-centered, marks mesh vertices that must remain fixed in space (unless the entire mesh is translated in space); significant at physical vertices only;
  - is set =1 for all physical block corners at intersection of fixed in space boundaries — i.e. boundaries with `ibc` = 1, 3, 4 or 8;
  - is set =1 at all physical vertices along fixed in space boundaries with `ibc` = 1, 3, 4 or 8, for which tangential rezoning was forbidden by setting `ifnotr12bc(ib,iblk)=.true.`;

- flag `lvx`, mask `msklvx`:
  - flag `lvx` is vertex-centered, marks strictly Lagrangian mesh vertices that move with the fluid and are not subject to rezoning; significant at physical vertices only;
  - is set =1 at all physical block corners that do not lie on any of the interblock boundaries with `ibc=5`, or are 3-vd or 3-dd block meeting points, and do not have flag `fix` set on;
  - is set =1 at all physical vertices along a physical edge with `ibc=0` (a center-of-convergence type of boundary);
  - is set =1 at all vertices (physical + ghost) where two rezoning directions are simultaneously indicated with the flags `tr1` and `tr2`.

- is set =1 at all physical vertices along moving boundaries with `ibc` = 2, 6 or 9, for which tangential rezoning was forbidden by setting `ifnotr12bc(ib,iblk)=.true.`;

- flag `ivx`, mask `mskivx`:
  - flag `ivx` is vertex-centered, marks physical vertices on Lagrangian (material) interfaces (as well as on the inflow/outflow boundaries!) that can only be tangentially rezoned along these interfaces; significant at physical vertices only; for any single vertex only one of the three flags `fix`, `lvx` or `ivx` is allowed to be set on;
  - is set =1 at all physical vertices that are not marked as strictly Lagrangian with `lvx=1` but have either `tr1=1` or `tr2=1`.

- flag `lf1`, mask `msklf1`:
  - flag `lf1` is face-centered, marks cell faces along mesh direction 1 across which no fluxing of material is allowed; significant on physical faces only;
  - is set =1 on all PR-faces (i.e. physical cell faces + those that protrude into the belt of ghost cells) along mesh direction $m = 1$ that separate cells with different materials;
  - is set =1 on all physical faces along a center-of-convergence boundary (i.e. one with `ibc=0`) along mesh direction $m = 1$.

- flag `lf2`, mask `msklf2`:
  - same as flag `lf1` but for mesh direction 2.

- flag `tr1`, mask `msktr1`:
  - flag `tr1` is vertex-centered, marks vertices that can only be tangentially rezoned along mesh direction 1; significant at physical vertices only;
  - is set =1 at both ends of all PR-faces along mesh direction $m = 1$ that have `lf1=1`;
  - is set =1 at all physical vertices on an inflow/outflow [i.e. with `(ibc=3.or.ibc=4)`] boundary along mesh direction $m = 1$.

- flag `tr2`, mask `msktr2`:
  - same as flag `tr1` but for mesh direction 2.

Note that it is important for all the ghost cells across "Lagrangian" boundaries (i.e. with `ibc` = 1, 2, 6, 8 or 9) to have the "unphysical" ("vacuum") material number `matnum` = 0. As a consequence of the above setting rules, **every physical vertex along a block edge, which does not lie on an interblock boundary with `ibc=5`, is marked with either `fix=1`, or `lvx=1`, or `ivx=1`; 3-vd and 3-dd block meeting corners are marked with either `fix=1` or `lvx=1`.** Figures 7.1 and 7.2 give two examples of flag setting on a single-block and a three-block meshes.

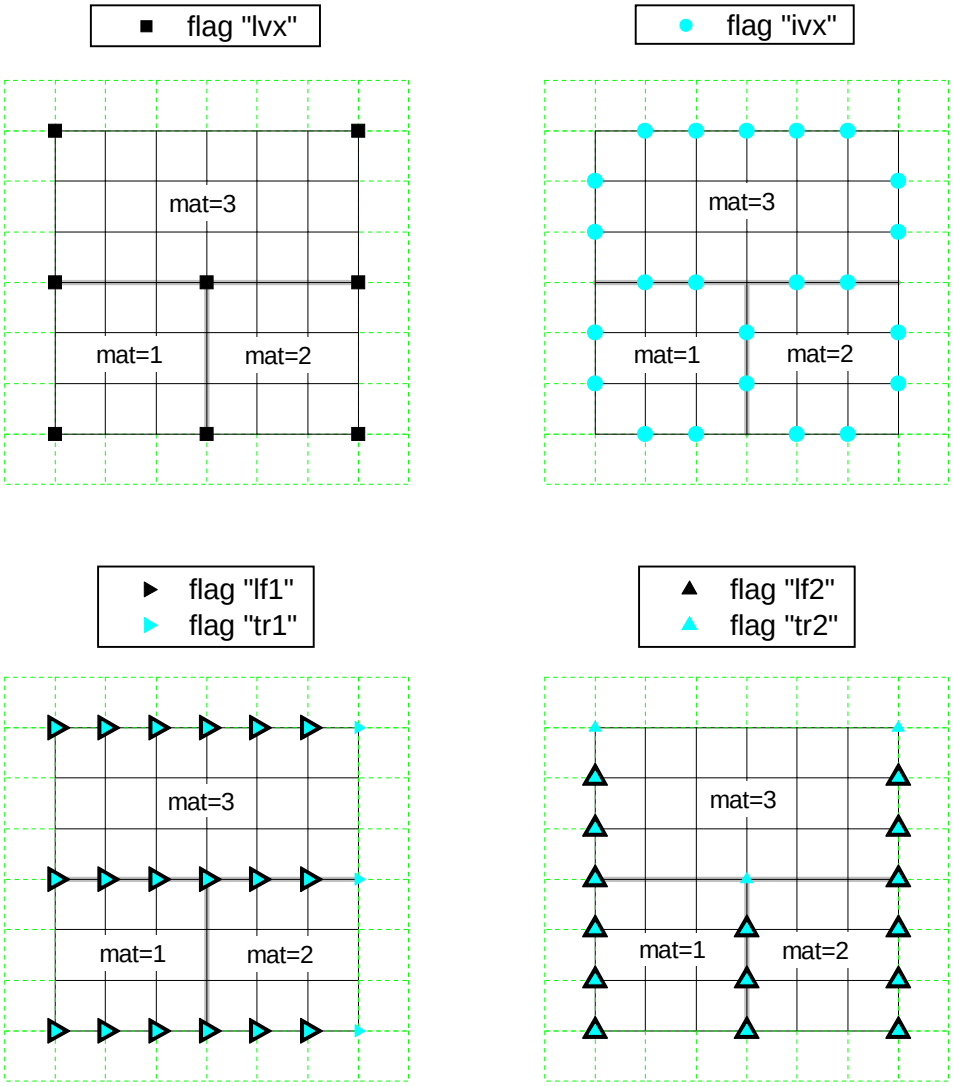


FIG. 7.1: Example of flag setting in a single-block mesh with 3 materials.

7.2. Layout of the subroutine `FLAGSET`

Step 1:

1. Initialize all flags to zero, `iflg=0`.
2. Set flag `mat=matnum` in all physical cells, part by part.
3. Set flags `gst=1` and `bnd=ib` in all ghost cells (1st + 2nd rows).
4. Set flag `mat=matnum` in all ghost cells (1st + 2nd rows) along the inflow/outflow boundaries with `(ibc=3.or.ibc=4)` by copying from the neighboring cells of the same block.

Step 2:

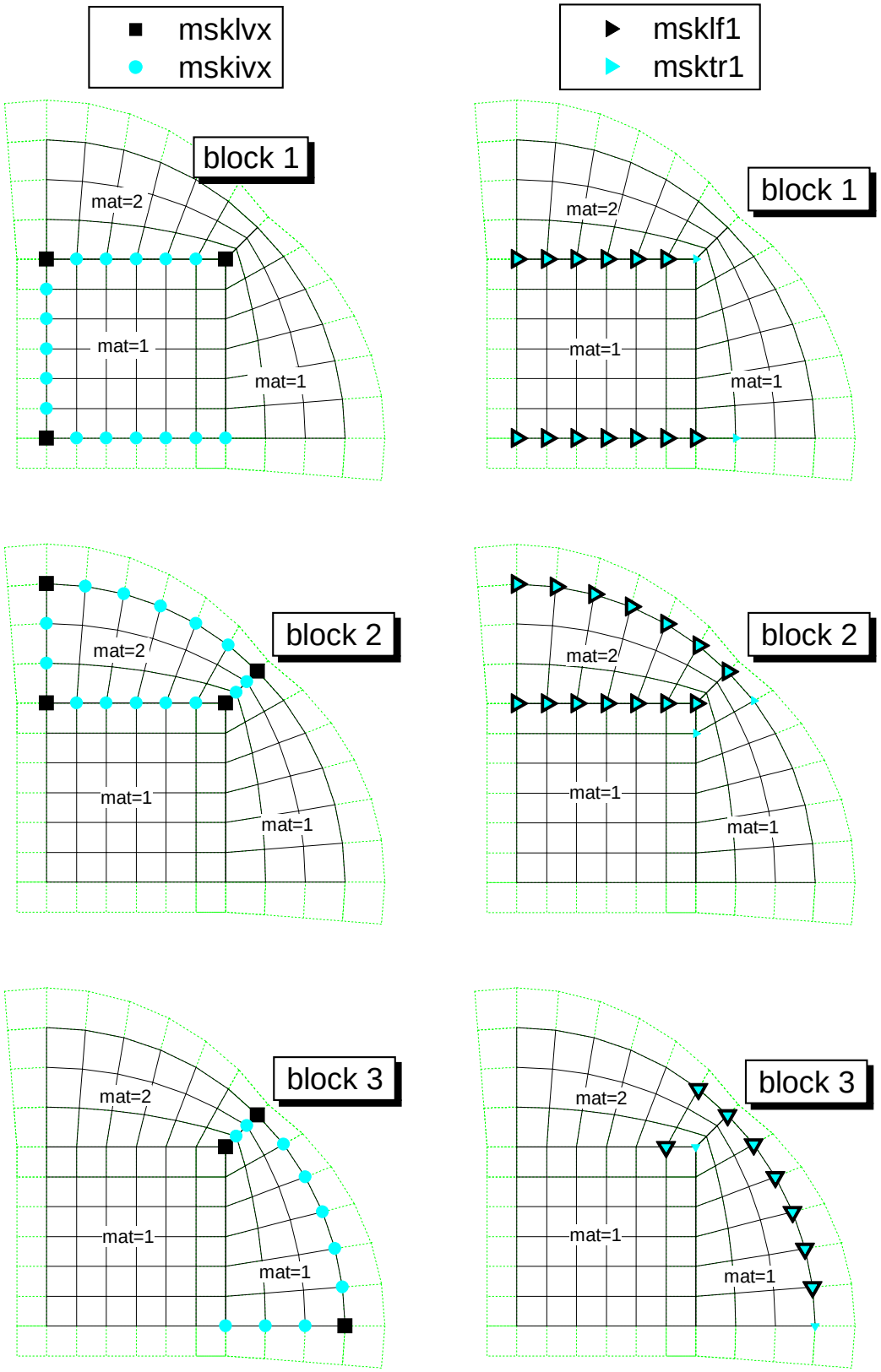


FIG. 7.2: Example of flag setting in a three-block mesh with 2 materials and a 3bk meeting point.

Set flag `mat=matnum` of the corresponding donor cell for all ghost cells (corner + non-corner) in the 1st row along all the interblock (i.e. with `ibc=5`) boundaries.

Step 3:

1. Set flag `cnv=1` for physical block corners.

2. Set flag `fix=1` for all physical block corners at intersections of fixed in space physical boundaries with `ibc` = 1, 3, 4 or 8 (including the 3-vd and 3-dd corners); for corners not satisfying this latter condition and not lying on interblock (i.e. with `ibc=5`) boundaries (or being 3-vd or 3-dd block meeting corners) set flag `lvx=1`.

Step 4:

1. Set flag `lf1(lf2)=1` for all PR-faces along mesh direction $m = 1(2)$ (i.e. physical cell faces + those that protrude into the belt of ghost cells) which separate cells with different materials; set flag `tr1(tr2)=1` for both end vertices of such faces.

2. Set flag `tr1(tr2)=1` at physical vertices along the inflow/outflow boundaries [i.e. with `(ibc=3.or.ibc=4)` and mesh direction $m = 1(2)$] — which are not material interfaces!

3. Set flag `lvx=1` at physical vertices, and `lf1(lf2)=1` at physical faces along a center-of-convergence boundary with `ibc=0` along mesh direction $m = 1(2)$.

4. Set flag `lvx=1` at all vertices (physical + ghost) where the flags `tr1` and `tr2` are both 'on'.

Step 5:

1. Set flag `fix=1` for all physical vertices along fixed in space boundaries with `ibc` = 1, 3, 4 or 8, for which tangential rezoning was forbidden by setting `ifnotr12bc(ib,iblk)=.true.`.

2. Set flag `lvx=1` for all physical vertices along moving physical boundaries with `ibc` = 2, 6 or 9, for which tangential rezoning was forbidden by setting `ifnotr12bc(ib,iblk)=.true.`.

Step 6:

Set flag `lvx=1` for all vertices that lie in mesh parts excluded by the user from rezoning by assigning `iflagprt(ip,iblk)=.true.`, and which do not have `fix=1`.

Step 7:

Set flag `ivx=1` at all physical vertices that are not marked with `lvx=1` (strictly Lagrangian) or `fix=1` (fixed in space) but lie on a material interface (boundaries with vacuum included), i.e. have either `tr1=1` or `tr2=1`.

Step 8:

Load array `ivx_(i)` of block-local indices for all `ivx`-vertices, and array `nvx_(iblk)`.

Step 9:

Set flag `gcc=1` for all ghost corner cells.

Step 10:

Load array `MBLKIV(I)`, which provides the block number for any cell with a global index `I`.

Step 11:

Set flag `bov=1` for all physical vertices along all block edges.

Step 12:

Set flag `pag=1` for all physically associated ghost cells (PAGC).

8. GEOMETRIC QUANTITIES

The geometric quantities given below are calculated in the `subroutine GEOM`. The primary variables which define the mesh are the vertex coordinates $\vec{x}_i = (x_i, y_i)$. Here index i combines the two mesh indices (i, j) along directions $m = 1$ and $m = 2$ respectively; face im is the edge of mesh cell i in mesh direction m; see Fig. 8.1.

For the neighbor mesh vertices we use the following index convention: i_{1+} is the vertex which lies next to vertex i along direction 1; i_{2+} is the vertex lying next to vertex i along direction 2; i_{12+} is the vertex lying next to vertex i along both direction 1 and 2 (across the diagonal of cell i); i_{1-} is the vertex preceding vertex i along direction 1; i_{2-} is the vertex preceding vertex i along direction 2. In the double-indexing convention, where $i = (i, j)$, we have $i_{1+} = (i+1, j)$, $i_{2+} = (i, j+1)$, $i_{12+} = (i+1, j+1)$, $i_{1-} = (i-1, j)$, $i_{2-} = (i, j-1)$. Then face $i1$ is the line segment between vertices i and i_{1+}; face $i2$ is the line segment between vertices i and i_{2+}.

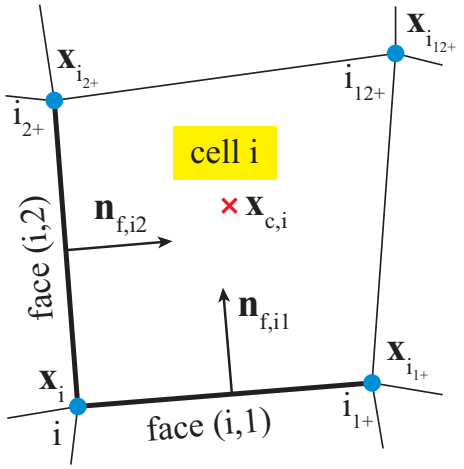


FIG. 8.1: Grid notation in a quadrilateral cell.

The length λ_{im} of face im is calculated as

$$\lambda_{im} = \sqrt{(x_{i_{m+}} - x_i)^2 + (y_{i_{m+}} - y_i)^2}. \tag{8.1}$$

The components of the unit normal vectors $\vec{n}_{f,im}$ are

$$\begin{aligned} \vec{n}_{f,i1} &= \left\{\lambda_{im}^{-1}(y_i - y_{i_{1+}}),\ \lambda_{im}^{-1}(x_{i_{1+}} - x_i)\right\}, \\ \vec{n}_{f,i2} &= \left\{\lambda_{im}^{-1}(y_{i_{2+}} - y_i),\ \lambda_{im}^{-1}(x_i - x_{i_{2+}})\right\}. \end{aligned} \tag{8.2}$$

The area S_{im} of face im (not to be mixed with its length λ_{im}) is given by

$$S_{im} = \int\limits_{\text{face } im} R\, d\lambda = \frac{1}{2}\left(R_i + R_{i_{m+}}\right)\lambda_{im}. \tag{8.3}$$

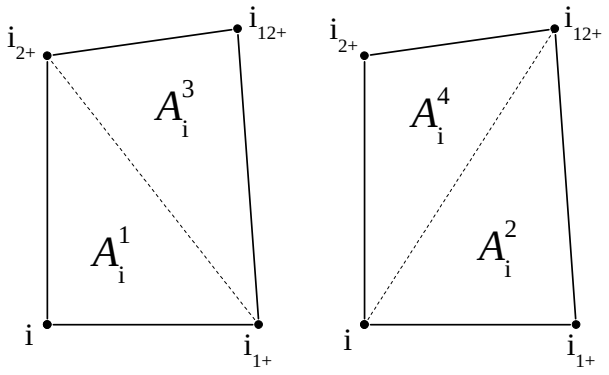


FIG. 8.2: Splitting of the mesh quadrilateral into two constituent triangles.

To evaluate any integral of the form $\int\limits_{A_i} \Phi(x,y)\, dxdy$ over the area A_i of cell i, where the integrand $\Phi(x,y)$ is defined at mesh nodes (x_i, y_i), we use a quadrature formula

$$\int\limits_{A_i} \Phi(x,y)\, dxdy = \frac{1}{6}\left[\Phi_i\left(A_i + A_i^1\right) + \Phi_{i_{1+}}\left(A_i + A_i^2\right) + \Phi_{i_{12+}}\left(A_i + A_i^3\right) + \Phi_{i_{2+}}\left(A_i + A_i^4\right)\right], \tag{8.4}$$

where A_i^α ($\alpha = 1,2,3,4$) are the areas of the constituent triangles in two possible triangulations of the quadrilateral i; see Fig. 8.2. Note that

$$A_i = A_i^1 + A_i^3 = A_i^2 + A_i^4. \tag{8.5}$$

The formula (8.4) is obtained by splitting the cell quadrilateral into two triangles (as shown in Fig. 8.2), applying to each triangle the quadrature formula

$$\int\limits_{\Delta} \Phi(x,y)\, dxdy = \frac{1}{3}\left(\Phi_1 + \Phi_2 + \Phi_3\right)\, A_\Delta, \tag{8.6}$$

and subsequent averaging over the two possible triangulations. Because formula (8.6) is exact for any linear function $\Phi = a + b\vec{x}$, formula (8.4) is also exact for any such function. It is also important that this formula is symmetric with respect to the 4 vertices of each mesh quadrilateral. On an orthogonal mesh we have $A_i^\alpha = \frac{1}{2}A_i$, and Eq. (8.4) simplifies to an obvious expression

$$\int\limits_{A_i} \Phi(x,y)\, dxdy = \frac{1}{4}\left(\Phi_i + \Phi_{i_{1+}} + \Phi_{i_{12+}} + \Phi_{i_{2+}}\right) A_i. \tag{8.7}$$

The areas of constituent triangles are given by

$$\begin{aligned} A_i^1 &= \tfrac{1}{2}\left(\vec{x}_{i_{1+}} - \vec{x}_i\right) \times \left(\vec{x}_{i_{2+}} - \vec{x}_i\right), \\ A_i^2 &= \tfrac{1}{2}\left(\vec{x}_{i_{12+}} - \vec{x}_{i_{1+}}\right) \times \left(\vec{x}_i - \vec{x}_{i_{1+}}\right), \\ A_i^3 &= \tfrac{1}{2}\left(\vec{x}_{i_{2+}} - \vec{x}_{i_{12+}}\right) \times \left(\vec{x}_{i_{1+}} - \vec{x}_{i_{12+}}\right), \\ A_i^4 &= \tfrac{1}{2}\left(\vec{x}_i - \vec{x}_{i_{2+}}\right) \times \left(\vec{x}_{i_{12+}} - \vec{x}_{i_{2+}}\right). \end{aligned} \tag{8.8}$$

These areas are all positive for any convex cell quadrilateral in a right-handed mesh.

The cell volume V_i is a particular case of integral (8.4), with $\Phi = R$ being either x or y. In this case we can simplify Eq. (8.4) to

$$V_i = \int\limits_{A_i} R\,dx\,dy = \frac{1}{3}\left(R_{i_{1+}} + R_{i_{2+}} + R_i\right) A_i^1 + \frac{1}{3}\left(R_{i_{1+}} + R_{i_{2+}} + R_{i_{12+}}\right) A_i^3. \tag{8.9}$$

If a physical cell with $V_i < 0$ is found on a right-handed mesh, this is diagnosed as a mesh crash and computation is aborted. Note that all the algebraic expressions in Eqs. (8.1)–(8.3), (8.8), (8.9) are exact on an arbitrary quadrilateral mesh in both the Cartesian (x, y) and the cylindrical (r, z) geometries.

The mean cylindrical radius of cell i is defined as

$$R_{av,i} = \frac{V_i}{A_i}. \tag{8.10}$$

The geometrical center of cell i is calculated as

$$\vec{x}_{c,i} = \frac{1}{4}\left(\vec{x}_i + \vec{x}_{i_{1+}} + \vec{x}_{i_{2+}} + \vec{x}_{i_{12+}}\right). \tag{8.11}$$

Correspondence with the code variables:

λ_{im}	= `flngth(i,m)`	length of face im;
$n_{f,i,m,x}$	= `fn(i,m,1)`	x-component of the unit normal to face im;
$n_{f,i,m,y}$	= `fn(i,m,2)`	y-component of the unit normal to face im
S_{im}	= `fa(i,m)`	area of face im;
A_i^α	= `a3angl(i,`α`)`	area of a constituent triangle α $(= 1, 2, 3, 4)$ of cell i;
V_i	= `vol(i)`	volume of cell i;
$x_{c,i}$	= `xc(i,1)`	x-component of the geometrical cell center;
$y_{c,i}$	= `xc(i,2)`	y-component of the geometrical cell center;
$R_{av,i}$	= `rav(i)`	average cylindrical radius of cell i;

---

9. MESH LIBRARY

There are a number of preprogrammed options for mesh construction in the RALEF code. They are distinguished by the value of parameter `igeom`. Some of these options (usually with single-digit values of `igeom`) are more general than the others. Each type of mesh has a number of free parameters, that can usually be specified via the `namelist/input/`, and, possibly, several fixed parameters that are assigned automatically. Below all the principal parameters, which control the mesh properties, are divided into three groups:

- **fixed:** these are the parameters, for which there is no freedom of choice in this particular mesh option;

- **user-must:** these are the parameters that *must* be specified by the user in this particular mesh option; for these parameters there are no default values;
- **user-can:** the user may either set new values of these parameters or stay by their default values; the corresponding default values are indicated.

9.1. `igeom=1` or 2: a multi-block rectangular x-y or r-z mesh

In this mesh option the number of blocks `nblks` is arbitrary, provided that `nblks` $\leq$ `nb`. Each mesh block is a rectangle. "x-y mesh" means that in each block mesh direction 1 is along the global x-axis, and mesh direction 2 is along the global y-axis, which is perpendicular to the x-axis. For `iradial=1` we have a rectangular r-z mesh, with r being identical with x. For `iradial=2` we have a rectangular r-z mesh, with r being identical with y.

For `igeom=1` the value of `iradial` is fixed at `iradial=0` (historically). For `igeom=2` the value of `iradial` is free to choose. An example of a single-block progressive (x, y) mesh with 2 parts along the x-axis is shown in Fig. 9.1.

Mesh parameters for `igeom=1`, uniform mesh:

- **fixed:** `iradial=0`
- **user-must:** `ncell(iprt,m,iblk)`
  `dx(iprt,m,iblk)`
- **user-can:**

`nblks`	def $= 1$
`nprts(m,iblk)`	def $= 1$
`x0(m,iblk)`	def $= (0.0, 0.0)$
`ibc(ib,iblk)`	def $= 2$
`nbc(k,ib,iblk)`	def $= 0$
`ghwidth`	def $= 10^{-4}$

Mesh parameters for `igeom=1`, progressive mesh:

- **fixed:** `iradial=0`
- **user-must:** `ncell(iprt,m,iblk)`
  `xxl(iprt,m,iblk)`
- **user-can:**

`nblks`	def $= 1$
`nprts(m,iblk)`	def $= 1$
`fdx(iprt,m,iblk)`	def $= 1.0$
`ibc(ib,iblk)`	def $= 2$
`nbc(k,ib,iblk)`	def $= 0$
`ghwidth`	def $= 10^{-4}$

Mesh parameters for `igeom=2`, uniform mesh:

- **fixed:** none
- **user-must:** `ncell(iprt,m,iblk)`
  `dx(iprt,m,iblk)`

- **user-can:** `iradial` def $= 0$
  `nblks` def $= 1$
  `nprts(m,iblk)` def $= 1$
  `x0(m,iblk)` def $= (0.0, 0.0)$
  `ibc(ib,iblk)` def $= 2$
  `nbc(k,ib,iblk)` def $= 0$
  `ghwidth` def $= 10^{-4}$

Mesh parameters for `igeom=2`, progressive mesh:

- **fixed:** none
- **user-must:** `ncell(iprt,m,iblk)`
  `xxl(iprt,m,iblk)`
- **user-can:** `iradial` def $= 0$
  `nblks` def $= 1$
  `nprts(m,iblk)` def $= 1$
  `fdx(iprt,m,iblk)` def $= 1.0$
  `ibc(ib,iblk)` def $= 2$
  `nbc(k,ib,iblk)` def $= 0$
  `ghwidth` def $= 10^{-4}$

Explanations:

- `nprts(m,iblk)` is the number of parts along mesh direction `m` in part `iprt` of block `iblk`;
- `ncell(iprt,m,iblk)` is the number of physical cells along mesh direction `m` in part `iprt` of block `iblk`;
- `x0(1,iblk)` is the x coordinate of the lower left corner (corner 1) of block `iblk`;
- `x0(2,iblk)` is the y coordinate of the lower left corner (corner 1) of block `iblk`;
- `dx(iprt,m,iblk)` is the cell size along mesh direction `m` (i.e. either along the global x-axis or along the global y-axis) in part `iprt` of block `iblk`;
- `xxl(iprt,1,iblk)` and `xxl(iprt+1,1,iblk)` are the x coordinates of the left and right bounds of part `iprt` in block `iblk`;
- `xxl(iprt,2,iblk)` and `xxl(iprt+1,2,iblk)` are the y coordinates of the lower and upper bounds of part `iprt` in block `iblk`;
- `fdx(iprt,m,iblk)` is the ratio of successive cell sizes along mesh direction `m` in part `iprt` of block `iblk`; "good" values must be not too far from `fdx(iprt,m,iblk)` $= 1.0$, say, within the interval $0.9 \lesssim$ `fdx(iprt,m,iblk)` $\lesssim 1.1$;
- `ghwidth` is the relative width of the ghost-cell bands;

A progressive-mesh option is chosen automatically when at least one value of `xxl(iprt,m,iblk)` is set different from its default value of `undef` $= -123456789.0$. In a progressive mesh, the values of `x0(m,iblk)` and `dx(iprt,m,iblk)` are calculated from the values of `xxl(iprt,m,iblk)` and `fdx(iprt,m,iblk)`; in particular, `x0(1,iblk)= xxl(1,1,iblk)`, `x0(2,iblk)= xxl(1,2,iblk)`.

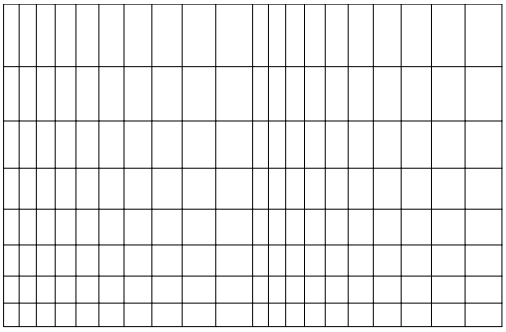


FIG. 9.1: A progressive (x, y) mesh with 2 parts along the x-axis in a single block.

9.2. `igeom=3` or `4`: a multi-block polar r-θ mesh

In this mesh option each block is a circular sector, with $x_2 = r$ being the radius, and $x_1 = \theta$ being the polar angle. The number of blocks `nblks` can be arbitrary, provided that `nblks` $\leq$ `nb`. the polar angle θ is measured from the vertical y-axis in degrees of arc, so that $x = r \sin\theta$, $y = r \cos\theta$. Hence, the mesh parameters `x0(1,iblk)`, `dx(iprt,1,iblk)`, and `xxl(iprt,1,iblk)` must be given in degrees.

For `igeom=3` the value of `iradial` is fixed at `iradial=0` (historically). For `igeom=4` the value of `iradial` is free to choose between 0, 1 and 2. Figure 9.2 shows an example of a single-block polar mesh, consisting of 3 parts along the $x_1 = \theta$ axis.

Mesh parameters for `igeom=3`, uniform mesh:

- **fixed:** `iradial=0`
- **user-must:** `ncell(iprt,m,iblk)`
  `dx(iprt,m,iblk)`
- **user-can:**

`nblks`	def $= 1$
`nprts(m,iblk)`	def $= 1$
`x0(m,iblk)`	def $= (0.0, 0.0)$
`ibc(ib,iblk)`	def $= 2$
`nbc(k,ib,iblk)`	def $= 0$
`ghwidth`	def $= 10^{-4}$

Mesh parameters for `igeom=3`, progressive mesh:

- **fixed:** `iradial=0`
- **user-must:** `ncell(iprt,m,iblk)`
  `xxl(iprt,m,iblk)`
- **user-can:**

`nblks`	def $= 1$
`nprts(m,iblk)`	def $= 1$
`fdx(iprt,m,iblk)`	def $= 1.0$
`ibc(ib,iblk)`	def $= 2$
`nbc(k,ib,iblk)`	def $= 0$
`ghwidth`	def $= 10^{-4}$

Mesh parameters for `igeom=4`, uniform mesh:

- **fixed:** none
- **user-must:** `ncell(iprt,m,iblk)`
  `dx(iprt,m,iblk)`
- **user-can:**

`iradial`	def $= 0$
`nblks`	def $= 1$
`nprts(m,iblk)`	def $= 1$
`x0(m,iblk)`	def $= (0.0, 0.0)$
`ibc(ib,iblk)`	def $= 2$
`nbc(k,ib,iblk)`	def $= 0$
`ghwidth`	def $= 10^{-4}$

Mesh parameters for `igeom=4`, progressive mesh:

- **fixed:** none
- **user-must:** `ncell(iprt,m,iblk)`
  `xxl(iprt,m,iblk)`
- **user-can:**

`iradial`	def $= 0$
`nblks`	def $= 1$
`nprts(m,iblk)`	def $= 1$
`fdx(iprt,m,iblk)`	def $= 1.0$
`ibc(ib,iblk)`	def $= 2$
`nbc(k,ib,iblk)`	def $= 0$
`ghwidth`	def $= 10^{-4}$

Explanations:

- `nprts(m,iblk)` is the number of parts along mesh direction `m` in part `iprt` of block `iblk`;
- `ncell(iprt,m,iblk)` is the number of physical cells along mesh direction `m` in part `iprt` of block `iblk`;
- `x0(1,iblk)` is the θ coordinate (in degrees) of the lower left corner (corner 1) of block `iblk`;
- `x0(2,iblk)` is the r coordinate of the lower left corner (corner 1) of block `iblk`;
- `dx(iprt,m,iblk)` is the cell size along mesh direction `m` [i.e. either along the θ-axis (`m=1`) or along the r-axis (`m=2`)] in part `iprt` of block `iblk`;
- `xxl(iprt,1,iblk)` and `xxl(iprt+1,1,iblk)` are respectively the θ coordinates (in degrees) of the left and right bounds of part `iprt` in block `iblk`;
- `xxl(iprt,2,iblk)` and `xxl(iprt+1,2,iblk)` are respectively the r coordinates of the lower and upper bounds of part `iprt` in block `iblk`;

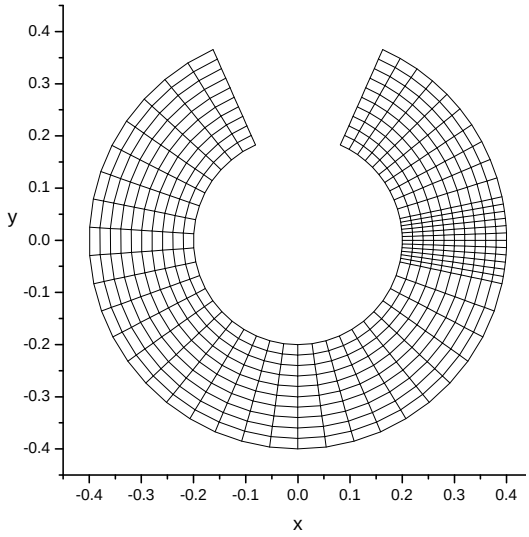


FIG. 9.2: A uniform (r, θ) mesh with 3 parts along the $x_1 = \theta$ axis in a single block.

– `fdx(iprt,m,iblk)` is the ratio of successive cell sizes along mesh direction `m` in part `iprt` of block `iblk`; "good" values must be not too far from `fdx(iprt,m,iblk)` $= 1.0$, say, within the interval $0.9 \lesssim$ `fdx(iprt,m,iblk)` $\lesssim 1.1$;

– `ghwidth` is the relative width of the ghost-cell bands;

The progressive-mesh option is chosen automatically when at least one value of `xxl(iprt,m,iblk)` is set different from its default value of `undef` $= -123456789.0$. In a progressive mesh, the values of `x0(m,iblk)` and `dx(iprt,m,iblk)` are calculated from the values of `xxl(iprt,m,iblk)` and `fdx(iprt,m,iblk)`; in particular, `x0(1,iblk)= xxl(1,1,iblk)`, `x0(2,iblk)= xxl(1,2,iblk)`.

9.3. `igeom=41`: a "snaky" band-like mesh

1. General description

This mesh is intended for computational domains in the form of narrow curved bands — to simulate, for example, thin curved foils. It may consist of an arbitrary number of blocks `nblks`. Every subsequent block `iblk+1` is attached to the previous block `iblk` along the edge `ib` $= 1$ in the "upper" block `iblk+1`, and the edge `ib` $= 2$ in the "lower" block `iblk`. The corresponding values of the block communication arrays `ibc` and `nbc` are assigned automatically in the `subroutine MSHP41`. A "snaky" mesh always meanders along mesh direction 2 in all blocks. An example of a "snaky" mesh, composed of two blocks, is shown in Fig. 9.3.

A "snaky" mesh consists of individual segments along mesh direction 2. Each segment is represented by a single part `jprt` along mesh direction 2 in a current block `iblk`. Any given segment can be either straight or curved along a circular arc. More precisely, a "snaky" mesh can be thought of as built around a continuous reference curve — a "spinal chord", which consists of straight and circular-arc segments. The "spinal chord" may lie anywhere between the left and the right boundaries of the mesh band. In Fig. 9.3 the "spinal chord"

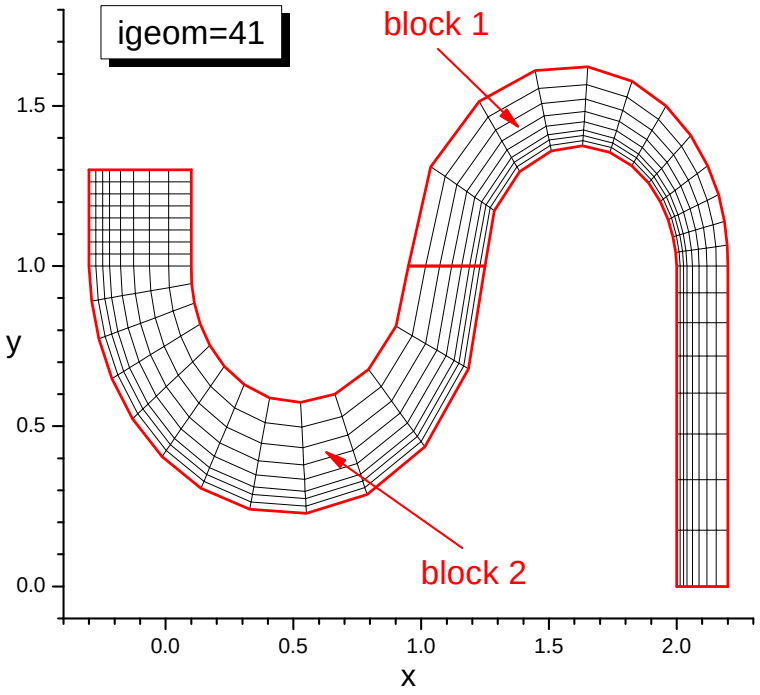


FIG. 9.3: A "snaky" mesh with 2 blocks; each block has 2 parts along each mesh direction.

passes through the middle of the mesh band. The global rotation angle Θ of the mesh band is defined as the rotation angle (in the counter-clockwise direction) of the local normal to the "spinal chord" with respect to the global x-axis. If the starting value $\Theta_0 = 0$, then the first mesh segment starts in the vertical direction along the global y-axis (as in Fig. 9.3).

The full set of user-defined parameters, which completely specify the "snaky" mesh and are to be assigned via the namelist `input` in file `'in2d'`, is as follows

- **user-must:**
```
ncell(1:nprts(1,1),1,1)
ncell(1:nprts(2,1),2,1:nblks)
dx(1:nprts(1,1),1,1)
dx(1:nprts(2,1),2,1:nblks)
```

- **user-can:**
```
iradial, nblks,
nprts(1:2,1:nblks) ,
xxl(1:nprts(2,iblk)+1,1:2,1:nblks),
fdx(1:nprts(m,iblk),1:2,1:nblks),
x0(1:2,1).
```

The parameters in the **user-must** group have no default values and must be specified by the user. The parameters in the **user-can** group have certain reasonable default values and are free to be left unspecified by the user.

2. The meaning of the mesh parameters

Number of cells

For mesh direction 1, the number of cells `ncell(ip,1,1)` must be prescribed for each part `ip` $= 1, 2, \ldots,$ `nprts(1,1)` but only in the first block. In all other blocks the same values are used. The mesh in Fig. 9.3 has `ncell(1,1,1)` $= 3$, `ncell(2,1,1)` $= 5$.

For mesh direction 2, one has to load the values of `ncell(jp,2,iblk)` for all parts `jp` $= 1, 2, \ldots,$ `nprts(2,iblk)` in all blocks `iblk` $= 1, 2, \ldots,$ `nblks`. The mesh in Fig. 9.3 has `ncell(1,2,1)` $= 8$, `ncell(2,2,1)` $= 12$, `ncell(1,2,2)` $= 12$, `ncell(2,2,2)` $= 8$.

Starting corner and edge

The coordinates (x_0, y_0) of the lower-left corner (corner # 1) in block 1 are given by the values of

$$\begin{aligned} x_0 &= \texttt{xxl(1,1,1)}, \\ y_0 &= \texttt{xxl(1,2,1)}. \end{aligned} \tag{9.1}$$

If no value is assigned to `xxl(1,1,1)` [and/or `xxl(1,2,1)`] in the input file '`in2d`', then the initial corner position is set to $x_0 = 0$ [and/or $y_0 = 0$]; in Fig. 9.3 we have $(x_0, y_0) = (2, 0)$.

The initial tilt angle Θ_0 of edge `ib` $= 1$ of block 1 is set by using the mesh parameter `x0(2,1)`:

$$\Theta_0 = \texttt{x0(2,1)}. \tag{9.2}$$

The default value is $\Theta_0 =$ `x0(2,1)` $= 0$.

Position of the "spinal chord"

The relative depth δ_{sc} of the "spinal chord" inside the mesh is defined by the parameter

$$0 \le \delta_{sc} = \texttt{x0(1,1)} \le 1. \tag{9.3}$$

In `subroutine MSHP41` the value of `x0(1,1)`, assigned by the user, is automatically trimmed to satisfy condition (9.3). If $\delta_{sc} = 0$ (the default value), the "spinal chord" coincides with the left boundary of the mesh band; if $\delta_{sc} = 1$, it coincides with the right boundary of the mesh band. In Fig. 9.3 the value $\delta_{sc} =$ `x0(1,1)` $= 0.5$ is used.

Dimensions

The principal dimensions of individual mesh segments are set by the `dx` array. The bottom width d_0 of the starting mesh segment, equal to the length of edge `ib` $= 1$ in block 1, is defined by the summed values of subarray `dx(1:nprts(1,1),1,1)`,

$$d_0 = \sum_{\texttt{ip}=1}^{\texttt{nprts(1,1)}} \texttt{dx(ip,1,1)}, \tag{9.4}$$

which are to be set for the block 1 only. More precisely,

`dx(ip,1,1)`

is the length of part `ip` along mesh direction 1 at the lower edge 1 in block 1. In Fig. 9.3 we have $d_0 = 0.2$.

The mesh extension along the "spinal chord" (i.e. along mesh direction 2) is defined by the subarray

`dx(1:nprts(2,1),2,1:nblks)`,

whose values must be assigned for all parts in all blocks along the mesh direction 2. The meaning of the elements of this subarray depends on whether the corresponding mesh segment is straight or circular:

- for a straight mesh segment, `dx(jp,2,iblk)` is the length of the corresponding part `jp` along mesh direction 2 in block `iblk`;
- for a circular mesh segment, `dx(jp,2,iblk)` is the turning angle $\Delta\Theta_{jp}$ in degrees of arc of the local normal to the "spinal chord" over the corresponding part `jp` of block `iblk`.

For straight segments `dx(jp,2,iblk)` must be positive, for circular segments `dx(jp,2,iblk)` can be both positive and negative. In Fig. 9.3 we have `dx(1,2,1)` $= 1$, `dx(2,2,1)` $= 180$, `dx(1,2,2)` $= -180$, `dx(2,2,2)` $= 0.3$.

Curvature radii and width scale factors

The curvature and the width of different mesh segments are controlled by the array `xxl(1:ns+1,1:2,1:nb)`. In contrast to the `dx` array, the values of `xxl(1:ns+1,1:2,1:nb)` refer to the interfaces between block parts along mesh direction 2, i.e. both the `xxl(jp+1,1,iblk)` and the `xxl(jp+1,2,iblk)` refer to the same interface between part `jp` and part `jp+1` along mesh direction 2 in block `iblk`.

The meaning of `xxl(jp+1,1,iblk)` is as follows:

- if `xxl(jp+1,1,iblk)` > 0, the mesh segment in the corresponding part `jp` along mesh direction 2 in block `iblk` is curved along a circular arc, with

$$R_{jp} = \texttt{xxl(jp+1,1,iblk)} \tag{9.5}$$

  being the radius of curvature of the "spinal chord" in this part;
- if `xxl(jp+1,1,iblk)` ≤ 0 (the default case), the mesh segment in the corresponding part `jp` along mesh direction 2 in block `iblk` is straight, and the value of `xxl(jp+1,1,iblk)` is not used; the default value is `xxl(jp+1,1,iblk)` $=$ `undef` $= -123456789$.

The mesh in Fig. 9.3 was constructed with `xxl(3,1,1)` $= 0.5$, `xxl(2,1,2)` $= 0.6$. Recall that the turning direction is defined by the sign of `dx(jp,2,iblk)`.

The value of `xxl(jp+1,2,iblk)` defines the stretch factor for the mesh width in the cross-section `jp+1` (i.e. between part `jp` and part `jp+1` along mesh direction 2 in block `iblk`). More precisely, the width d_{jp} of the mesh band at the cross-section `jp+1` is given by

$$d_{jp} = d_0 \cdot \texttt{xxl(jp+1,2,iblk)}, \tag{9.6}$$

where d_0 is the initial width of the starting edge, defined in Eq. (9.4). Inside a given part `jp`, the mesh width varies linearly along the "spinal chord" from d_{jp} at the bottom to d_{jp+1} at the top. The mesh in Fig. 9.3 was constructed with `xxl(3,2,1)` $= 1.5$, `xxl(2,2,2)` $=$ `xxl(3,2,2)` $= 2.0$.

Progression factors

The mesh progression factors `fdx(1:ns,1:2,1:nblks)` have their usual meaning: `fdx(ip,m,iblk)` is the ratio of successive cell sizes along mesh direction `m` in part `ip` of block `iblk`; the default value is `fdx(ip,m,iblk)` $= 1$. For the mesh direction 1, it is sufficient to assign the values `fdx(1:nprts(1,1),1,1)` in the first block only; the same values are used in all other blocks.

The mesh in Fig. 9.3 was constructed with the values `fdx(1:2,1,1)` $= 1.0, 1.2$, `fdx(1:2,2,1)` $= 0.9, 1.15$, `fdx(1:2,2,2)` $= 0.9, 1.0$.

9.4. `igeom=5` or `6`: a cylindrical (or spherical) mesh in a 180° semi-circle with a free-float center

This mesh consists of `nblks` = 4 blocks and represents a half-circle with a rectangle in the central region. It is shown in Fig. 9.4. The central rectangle has edge ratio 1:2, so that when the mesh is reflected with respect to the y-axis, it becomes a full circle with a square in the center. It has 4 main free parameters: two radii r_0 — the radius of the circumcircle around the central square, and r_{max} — the outer radius of the mesh.

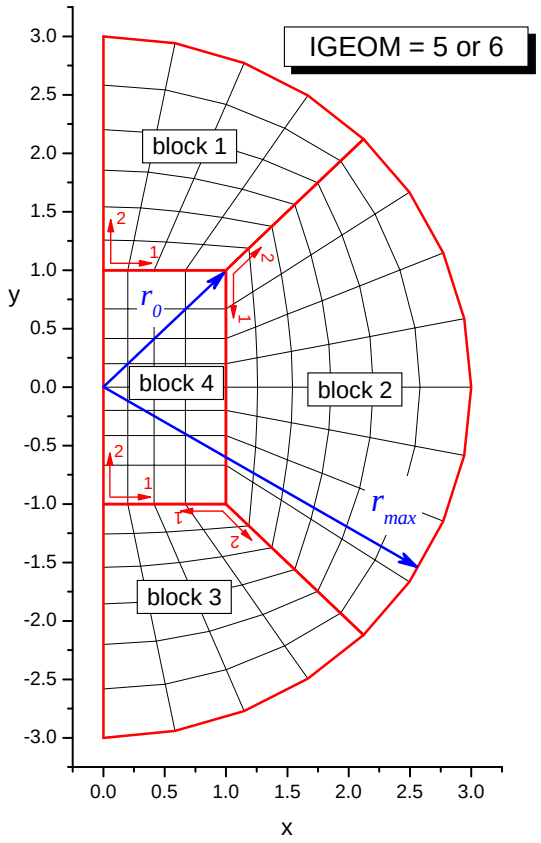


FIG. 9.4: A 180° 4-block cylindrical (or spherical) mesh with a free-float center.

Mesh parameters for `igeom=5`*:*

- **fixed:** `iradial=0`
  `nblks=4`
  `nprts(m,iblk) =1`
  `ncell(1,m,iblk)` for `iblk` ≥ 2
  `ibc(ib,iblk)` (except for `ibc(2,1)` and `ibc(3,1)`)
  `nbc(ib,iblk)`
- **user-must:** `ncell(1,1,1)`
  `ncell(1,2,1)`
  `x0(1,1)` $= r_0$
  `x0(2,1)` $= r_{max}$

- **user-can:** `ibc(2,1)` def $= 2$
  `ibc(3,1)` def $= 2$
  `ghwidth` def $= 10^{-4}$

Mesh parameters for `igeom=6` *(a full sphere):*

- **fixed:** `iradial=1`
  `nblks=4`
  `nprts(m,iblk) =1`
  `ncell(1,m,iblk)` for `iblk` ≥ 2
  `ibc(ib,iblk)` (except for `ibc(2,1)` and `ibc(3,1)`)
  `nbc(ib,iblk)`
- **user-must:** `ncell(1,1,1)`
  `ncell(1,2,1)`
  `x0(1,1)` $= r_0$
  `x0(2,1)` $= r_{max}$
- **user-can:** `ibc(2,1)` def $= 2$
  `ibc(3,1)` def $= 2$
  `ghwidth` def $= 10^{-4}$

Explanations:

- `ncell(1,1,1)` is the number of physical cells along the short edge of the central rectangle in block 4; its long edge has `2ncell(1,1,1)` mesh cells;
- `ncell(1,2,1)` is the number of physical cells along the radial direction in blocks 1, 2 and 3;
- `x0(1,1)` is the smaller radius r_0;
- `x0(2,1)` is the larger radius $r_{max} > r_0$;
- `ibc(2,1)` is the type of boundary condition along the circular part of the outer boundary; typically `ibc(2,1)=2`;
- `ibc(3,1)` is the type of boundary condition along the diameter part of the outer boundary; typically `ibc(2,1)=1`;

9.5. `igeom=22`: a multi-block arbitrary-quadrangle (AQ) x-y or r-z mesh

In this mesh option the number of blocks `nblks` is arbitrary, provided that `nblks` $\leq$ `nb`. Every mesh block is an arbitrary quadrangle. All mesh lines are straight. An AQ mesh is always assumed to be progressive along each quadrangle edge with independent common ratios.

Mesh parameters for `igeom=22`*:*

- **fixed:** none

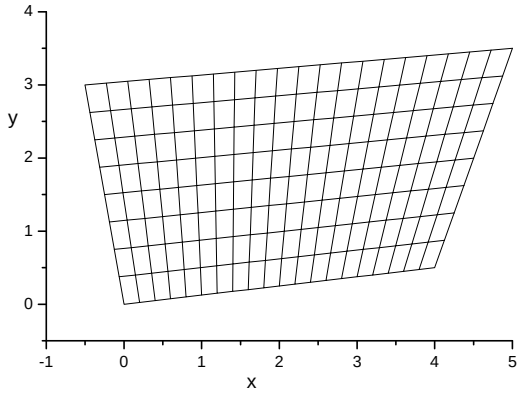


FIG. 9.5: An arbitrary-quadrangle mesh in a single block.

- **user-must:** `ncell(iprt,m,iblk)`
  `x0aq(m,ic,iblk)`
- **user-can:**

`iradial`	def $= 0$
`nblks`	def $= 1$
`nprts(m,iblk)`	def $= 1$
`fprtaq(iprt,ib,iblk)`	def $= 1.0$
`fdxaq(iprt,ib,iblk)`	def $= 1.0$
`ibc(ib,iblk)`	def $= 2$
`nbc(k,ib,iblk)`	def $= 0$
`ghwidth`	def $= 10^{-4}$

Explanations:

- `nblks` is the total number of mesh blocks;
- `nprts(m,iblk)` is the number of parts along mesh direction `m` in block `iblk`;
- `ncell(iprt,m,iblk)` is the number of physical cells along mesh direction `m` in part `iprt` of block `iblk`;
- `x0aq(1,ic,iblk)` is the x $(= x_1)$ coordinate of corner `ic` of block `iblk`; corners are numbered according to the CAVEAT convention, i.e. corner 1 is the lower-left one, corner 2 is the upper-right one, corner 3 is the upper-left one, corner 4 is the lower-right one;
- `x0aq(2,ic,iblk)` is the y $(= x_2)$ coordinate of corner `ic` of block `iblk`;
- `fprtaq(iprt,ib,iblk)` are the weights for part lengthes along edge `ib` in block `iblk`, i.e. `fprtaq(i,ib,iblk)` : `fprtaq(i+1,ib,iblk)` is the ratio of lengthes of parts `i` and `i+1` along edge `ib` in block `iblk`; normalization of `fprtaq(iprt,ib,iblk)` is arbitrary;
- `fdxaq(iprt,ib,iblk)` is the common ratio of successive cell sizes in part `iprt` of edge `ib` of block `iblk`; "good" values must be not too far from `fdxaq(iprt,ib,iblk)` $= 1.0$, say, within the interval $0.9 \lesssim$ `fdxaq(iprt,ib,iblk)` $\lesssim 1.1$;
- `ghwidth` is the relative width of the ghost-cell bands;

9.6. `igeom=23`: a multi-block polygon-mosaic mesh

In this option the number of blocks `nblks` is a free parameter. Every mesh block `iblk` is composed of `nprts(1,iblk)` $\times$ `nprts(2,iblk)` parts, where each part is an arbitrary quadrangle. Within each part the mesh lines are straight. Within any given block the mesh lines are piece-wise straight. In general, the lengthes of cell faces along every interface between different parts make a geometric progression with a common ratio, which is allowed to have individual values along different interfaces. In other words, each part of every block has an individual AQ mesh.

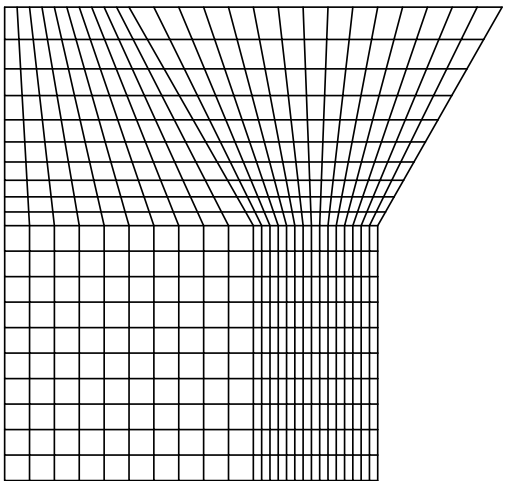


FIG. 9.6: An polygon mosaic mesh in a single block, which consists of two parts along each of the two mesh directions.

Mesh parameters for `igeom=23`:

- **fixed:** none
- **user-must:** `ncell(iprt,m,iblk)`
  `xxp(1:nprts(1,iblk)+1,1:nprts(2,iblk)+1,1:2,iblk)`
- **user-can:** `iradial`
  `nblks`
  `nprts(m,iblk)`
  `fdxp(1:nprts(1,iblk)+1,1:nprts(2,iblk)+1,1:2,iblk)`
  `ibc(ib,iblk)`
  `nbc(k,ib,iblk)`
  `ghwidth`

Explanations:

– `nblks` is the total number of mesh blocks;

– `nprts(m,iblk)` is the number of parts along mesh direction `m` in block `iblk`;

– `ncell(iprt,m,iblk)` is the number of physical cells along mesh direction `m` in part `iprt` (along this direction) of block `iblk`;

– `xxp(iprt,jprt,1,iblk)` is the x $(= x_1)$ coordinate of the lower-left corner of part `(iprt,jprt)` in block `iblk`; `iprt` is the part index along mesh direction 1, `jprt` is the part index along mesh direction 2; correspondingly, the lower-right, upper-left, and upper-right corners of part `(iprt,jprt)` in block `iblk` have the x-coordinates `xxp(iprt+1,jprt,1,iblk)`, `xxp(iprt,jprt+1,1,iblk)`, and `xxp(iprt+1,jprt+1,1,iblk)`; default value is `undef`;

– `xxp(iprt,jprt,2,iblk)` is the y $(= x_2)$ coordinate of the lower-left corner of part `(iprt,jprt)` in block `iblk`; default value is `undef`;

– `fdxp(iprt,jprt,1,iblk)` is the common ratio of successive cell sizes along the part edge, starting from the lower-left corner of part `(iprt,jprt)` along mesh direction 1, in block `iblk`; correspondingly, `fdxp(iprt,jprt+1,1,iblk)` is the common ratio of successive cell sizes along the part edge, starting from the upper-left corner of part `(iprt,jprt)` along mesh direction 1; default value is 1.0;

– `fdxp(iprt,jprt,2,iblk)` is the common ratio of successive cell sizes along the part edge, starting from the lower-left corner of part `(iprt,jprt)` along mesh direction 2, in block `iblk`; correspondingly, `fdxp(iprt+1,jprt,1,iblk)` is the common ratio of successive cell sizes along the part edge, starting from the lower-right corner of part `(iprt,jprt)` along mesh direction 2; default value is 1.0;

– `ghwidth` is the relative width of the ghost-cell bands;

9.7. `igeom=50`: a 5-block cylindrical mesh in a full 360° circle with a free-float center

This is a mesh for a cylindrical shell, originally designed for the wobbler problem to study the ϕ asymmetry of cylindrical implosions. The 4 outer blocks represent an annular region with a void in the center; the optional 5-th block covers the central region (see Fig. 9.7).

Mesh parameters for `igeom=50`, radially uniform mesh:

- **fixed:** `iradial=0`
  `nprts(m,iblk)` `=1` along ϕ direction
  `nprts(m,iblk)` along r direction for `iblk` ≥ 2
  `ncell(iprt,m,iblk)` for `iblk` ≥ 2
  `dx(iprt,1,iblk)`
  `ibc(ib,iblk)` (except for `ibc(2,1)`)
  `nbc(ib,iblk)`
- **user-must:** `ncell(1,1,1)`
  `ncell(iprt,2,1)`
  `x0(2,1)` $= r_0$
  `dx(iprt,2,1)`
- **user-can:** `nblks` $(= 4$ or $5)$ def $= 4$
  `nprts(2,1)` def $= 1$
  `ibc(2,1)` def $= 2$
  `ghwidth` def $= 10^{-4}$

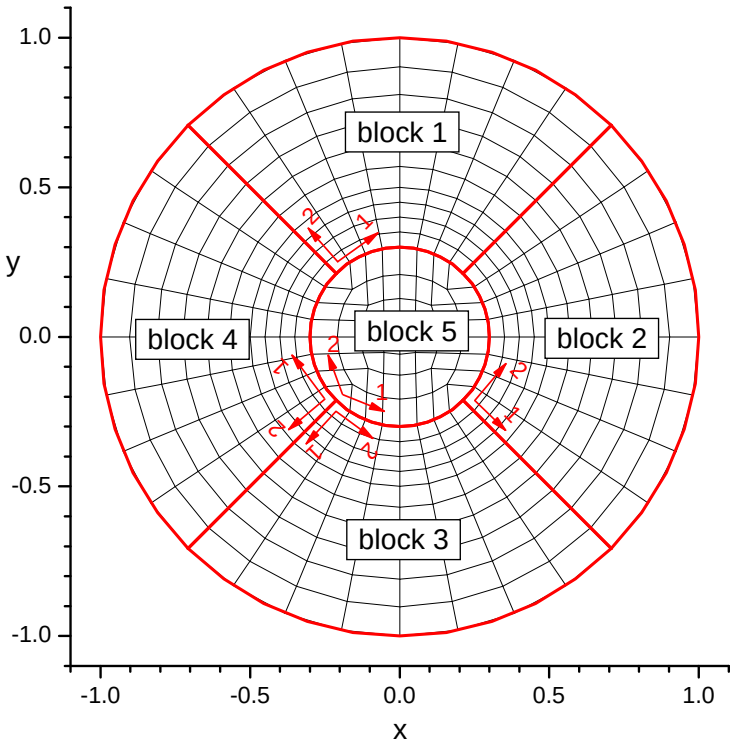


FIG. 9.7: A 360° full-circle 5-block cylindrical mesh with a free-float center.

Mesh parameters for `igeom=50`*, radially progressive mesh:*

- **fixed:** `iradial=0`
  `nprts(m,iblk)` `=1` along ϕ direction
  `nprts(m,iblk)` along r direction for `iblk` ≥ 2
  `ncell(iprt,m,iblk)` for `iblk` ≥ 2
  `xxl(1,m,iblk)` along ϕ direction
  `xxl(2,m,iblk)` along ϕ direction
  `fdx(iprt,1,iblk)=1`
  `ibc(ib,iblk)` (except for `ibc(2,1)`)
  `nbc(ib,iblk)`
- **user-must:** `ncell(1,1,1)`
  `ncell(iprt,2,1)`
  `xxl(1:nnprad+1,2,1)`
- **user-can:**

`nblks` (= 4 or 5)	def = 4
`nprts(2,1)` = *nnprad*	def = 1
`fdx(iprt,2,1)`	def = 1.0
`ibc(2,1)`	def = 2
`ghwidth`	def = 10^{-4}

Explanations:

– `nprts(2,1)` is the number of parts along the radial direction in each of the 4 outer blocks;

– `ncell(1,1,1)` is the number of physical cells along the azimuth ϕ in each of the 4 outer blocks (i.e. over the ϕ interval of 90°; it is also the number of cells along each edge of the central 5-th block;

– `ncell(iprt,2,1)` is the number of physical cells in part `iprt` along the radial direction in each of the 4 outer blocks;

– `x0(2,1)` is the radius $r_0 > 0$ of the central cavity (block 5);

– `dx(iprt,2,1)` is the cell size in part `iprt` along the radial direction in each of the 4 outer blocks;

– `xxl(iprt,2,1)` is the lower-end radius of part `iprt` along the radial direction in each of the 4 outer blocks;

– `fdx(iprt,2,1)` is the ratio of successive cell sizes in part `iprt` along the radial direction in each of the 4 outer blocks;

– `ibc(2,1)` is the type of boundary condition along the entire outer boundary; typically `ibc(2,1)=2`.

9.8. `igeom=51`: a multi-tier full-circle cylindrical mesh with a free-float center

1. General description

Here we construct a full-circle quasi-polar mesh with a free-float center, labeled as an H_4-mesh (a "Hohlraum" mesh extending over the 4 quadrants of the polar angle θ). It is only suitable for Cartesian geometry with `iradial` $= 0$. The core of the H_4-mesh consists of $n_t \geq 1$ full radial tiers. Each full tier extends over 360° and is comprised of 4 neighboring blocks. With one extra block in the center (which is chosen to be the block # 1), the total number of blocks in the mesh core is $4n_t + 1$.

Beside the $4n_t + 1$ core blocks, the H_4-mesh can have 1, 2, or 3 extra protruding blocks ("cap" blocks) in the unfilled $n_t + 1$-th tier, i.e. the total number of blocks in the H_4-mesh is

$$5 \leq 4n_t + 1 \leq \texttt{nblks} < 4n_t + 5. \tag{9.7}$$

Generally, different "cap" blocks must not touch one another along their side edges.

An example of the H_4-mesh with $n_t = 2$ full tiers and 2 extra "cap" blocks is shown in Fig. 9.8. The H_4-mesh is constructed in the progressive-mesh mode only, i.e. all the mesh dimensions are supposed to be set by assigning the arrays `xxl(ns+1,2,nb)` and `fdx(ns,2,nb)`; the values of the mesh parameters `x0(2,nb)` and `dx(ns,2,nb)` are not used.

The full set of parameters that completely specify the physical part of the H_4 mesh consists of the following variables and arrays

```
nblks,
nprts(1:2,1:nb),
ncell(1:ns,1:2,1:nb),
xxl(1:ns+1,1:2,1:nb),
fdx(1:ns,1:2,1:nb),
nbc(1:2,1:4,1:nb).
```
(9.8)

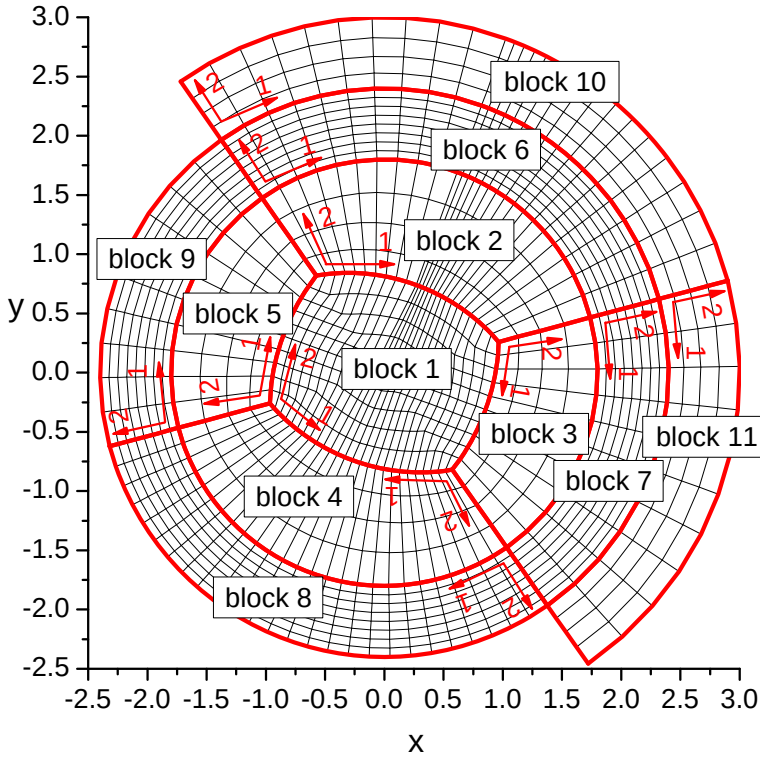


FIG. 9.8: A full-circle 11-block cylindrical mesh with a free-float center, 9 core blocks and 2 "cap" blocks (blocks 10 and 11).

The mesh directions $\mathtt{m} = 1$ and $\mathtt{m} = 2$ are rigidly fixed within every block: in all blocks other than the central block 1 mesh direction 1 is always along the polar θ angle (as measured from the vertical axis in the clockwise direction), while mesh direction 2 is always along the cylindrical radius r.

The mesh in the central block # 1 can be continuously stretched between two extreme configurations: a rectangle and a full circle. The amount of stretch is independent along the two perpendicular directions and is controlled by the values of the two parameters $0 \le \mathtt{fdx(1,1,1)} \le 1$ (the amount of stretch between blocks 4–1–2) and $0 \le \mathtt{fdx(1,2,1)} \le 1$ (the amount of stretch between blocks 5–1–3). For the default values of $\mathtt{fdx(1,1,1)} = \mathtt{fdx(1,2,1)} = 1$ we have a rectangular shape of the central block, for $\mathtt{fdx(1,1,1)} = \mathtt{fdx(1,2,1)} = 0$ — a circular one. The four corners of block # 1 and its diagonals do not participate in the stretch. The mesh in Fig. 9.8 has been constructed with the values of $\mathtt{fdx(1,1,1)} = \mathtt{fdx(1,2,1)} = 0.5$.

In practice, to construct the H_4 mesh, one does not need the full set of parameters (9.8) but only its certain limited subset. We call the subset of the full list (9.8) that is actually used for mesh construction the *principal* mesh parameters. As usual, some of the principal mesh parameters have default values and must not (but can) be specified by the user, others do not have default values and must be specified in the `namelist/input/`.

A special feature of the H_4 mesh is that every individual block can be divided into parts in an arbitrary manner, independently of how other blocks are divided into parts. In other words, the values of `nprts(m,iblk)` can be chosen independently for each mesh direction $\mathtt{m} = 1, 2$ in every block $\mathtt{iblk} = 1, 2, \ldots, \mathtt{nblks}$. But then, because contacting blocks must have the same number of cells along common edges, the user-defined values of

`ncell(iprt,m,iblk)` must satisfy certain consistency conditions.

2. Principal mesh parameters

igeom=51: mesh parameters for the θ-direction

Parameters used for the physical θ-mesh construction:

$$\left.\begin{array}{l}\texttt{nprts(1,iblk)},\\ \texttt{ncell(1:nprts(1,iblk),1,iblk)},\\ \texttt{xxl(1:nprts(1,iblk)+1,1,iblk)},\\ \texttt{fdx(1:nprts(1,iblk),1,iblk)},\end{array}\right\}\ \texttt{iblk} = 2 \to 5. \tag{9.9}$$

Parameters that must be specified in the `namelist/input/`:

$$\begin{array}{l}\texttt{ncell(1,1,2)},\\ \texttt{ncell(1,1,3)},\\ \texttt{xxl(1,1,2)},\\ \texttt{xxl(2,1,2)}.\end{array} \tag{9.10}$$

If more parameters are specified than in the "short" list (9.10), they must satisfy the following consistency conditions:

$$\begin{array}{l}\displaystyle\sum_j \texttt{ncell(j,1,4)} = \sum_i \texttt{ncell(i,1,2)},\\ \displaystyle\sum_j \texttt{ncell(j,1,5)} = \sum_i \texttt{ncell(i,1,3)};\end{array} \tag{9.11}$$

$$\left.\begin{array}{l}\texttt{xxl(1,1,iblk+1)} = \texttt{xxl(nprts(1,iblk)+1,1,iblk)},\\ \texttt{xxl(nprts(1,iblk+1)+1,1,iblk+1)} = \texttt{xxl(1,1,iblk)} + 180,\end{array}\right\}\ \texttt{iblk} = 2 \to 4. \tag{9.12}$$

$$\texttt{xxl(i,1,iblk)} < \texttt{xxl(i+1,1,iblk)}, \quad \texttt{iblk} = 2 \to 5. \tag{9.13}$$

Note that the values of `xxl(i,1,iblk)` for `iblk` = 2–5 specify the angular boundaries of the corresponding block parts in degrees of arc (as measured from the vertical axis in the clockwise direction). If only the values of `xxl(1:2,1,2)` are assigned in the `namelist/input/`, then the values of `xxl(1:2,1,iblk)` for `iblk` = 3–5 are calculated from the consistency conditions (9.12).

igeom=51: mesh parameters for the r-direction in the core blocks

Parameters used for the physical r-mesh construction:

$$\begin{array}{l}\left.\begin{array}{l}\texttt{nprts(2,iblk)},\\ \texttt{ncell(1:nprts(2,iblk),2,iblk)},\end{array}\right\}\ \texttt{iblk} = 2 \to 4n_t + 1;\\ \\ \left.\begin{array}{l}\texttt{xxl(1:nprts(2,iblk)+1,2,iblk)},\\ \texttt{fdx(1:nprts(2,iblk),2,iblk)},\end{array}\right\}\ \texttt{iblk} = 2, 6, \ldots, 4n_t - 2.\end{array} \tag{9.14}$$

Parameters that must be specified in the `namelist/input/`:

$$\left.\begin{array}{l}\texttt{ncell(1,2,iblk)},\\ \texttt{xxl(1,2,iblk)},\\ \texttt{xxl(2,2,}4n_t-2\texttt{)}.\end{array}\right\}\ \texttt{iblk} = 2, 6, \ldots, 4n_t - 2. \tag{9.15}$$

Consistency conditions:

$$\texttt{xxl(1,2,iblk+4)} = \texttt{xxl(nprts(2,iblk)+1,2,iblk)}, \quad \texttt{iblk} = 2, 3, \ldots, 4n_t - 6. \tag{9.16}$$

$$\texttt{xxl(i,2,iblk)} < \texttt{xxl(i+1,2,iblk)}, \quad \texttt{iblk} = 2, 6, \ldots, 4n_t - 2. \tag{9.17}$$

`igeom=51`: mesh parameters for the "cap" blocks

Parameters used for the physical θ-mesh construction:

$$\left.\begin{array}{l}\texttt{nprts(1,iblk)},\\ \texttt{ncell(1:nprts(1,iblk),1,iblk)},\\ \texttt{nbc(1,1,iblk)},\end{array}\right\}\ \texttt{iblk} = 4n_t + 2 \rightarrow \texttt{nblks}. \tag{9.18}$$

Parameters used for the physical r-mesh construction:

$$\left.\begin{array}{l}\texttt{nprts(2,iblk)},\\ \texttt{ncell(1:nprts(2,iblk),2,iblk)},\end{array}\right\}\ \texttt{iblk} = 4n_t + 2 \rightarrow \texttt{nblks};$$

$$\left.\begin{array}{l}\texttt{xxl(1:nprts(2,iblk)+1,2,iblk)},\\ \texttt{fdx(1:nprts(2,iblk),2,iblk)},\end{array}\right\}\ \texttt{iblk} = 4n_t + 2. \tag{9.19}$$

Parameters that must be specified in the `namelist/input/`:

$$\left.\begin{array}{l}\texttt{ncell(1,2,iblk)},\\ \texttt{xxl(2,2,iblk)},\end{array}\right\}\ \texttt{iblk} = 4n_t + 2;$$

$$\texttt{nbc(1,1,iblk)}, \quad \texttt{iblk} = 4n_t + 2, \ldots, \texttt{nblks}. \tag{9.20}$$

Consistency conditions:

$$\sum_j \texttt{ncell(j,1,iblk)} = \sum_i \texttt{ncell(i,1,jblk)},$$

$$\texttt{jblk} = \texttt{nbc(1,1,iblk)}, \quad \texttt{iblk} = 4n_t + 2 \rightarrow \texttt{nblks}; \tag{9.21}$$

$$\texttt{xxl(1,2,iblk+4)} = \texttt{xxl(nprts(2,iblk)+1,2,iblk)}, \quad \texttt{iblk} = 4n_t - 2. \tag{9.22}$$

$$\texttt{xxl(i,2,iblk)} < \texttt{xxl(i+1,2,iblk)}, \quad \texttt{iblk} = 4n_t + 2 \rightarrow \texttt{nblks}. \tag{9.23}$$

`igeom=51`: stretch parameters for the central block # 1

Parameters used to stretch the physical mesh in block # 1:

$$\begin{array}{l}0 \le \texttt{fdx(1,1,1)} \le 1,\\ 0 \le \texttt{fdx(1,2,1)} \le 1.\end{array} \tag{9.24}$$

Having specified the values of the above listed principal mesh parameters, one fully determines the physical part of the H_4 mesh. As usual, the relative width of the ghost-cell layer along the outer boundary is controlled by the user-defined parameter `ghwidth` (default value 10^{-4}). The type of the outer boundary condition is specified by the user-defined values of `ibc(ib,iblk)` along the outer block edges that border vacuum.

However, there remains freedom for dividing into parts those mesh blocks, whose values of `nprts(m,iblk)` and `ncell(ip,m,iblk)` are not required for mesh construction (like blocks 7–9 in Fig. 9.8. Hence, the user can, in addition, specify the values of these parameters in all the remaining blocks as well — provided that they are consistent with the number of cells along the common edges with other blocks. Such an additional division into parts can be used for complex distribution of different materials among different parts of any block in the mesh. The latter is facilitated by the fact that the mesh construction routine `MSHB51` does not change the values of `nprts(m,iblk)` and `ncell(ip,m,iblk)`. The mutual consistency of the values of these arrays is checked in the `subroutine MSHP51`, and the job is aborted if an inconsistency is found.

Finally, the assignment rules for the H_4-mesh parameters can be summarized as follows.

- **fixed or restricted:** `iradial=0`,
  `nblks` ≥ 5,
  `ibc(ib,iblk)`$=5$ at common block faces.
- **user-must:** `nblks`
  `ncell(1,1,2)`
  `ncell(1,1,3)`
  `xxl(1:2,1,2)`
  `ncell(1,2,2)`, `ncell(1,2,6)`, ...
  `xxl(1,2,2)`, `xxl(1,2,6)`, ...
  `xxl(2,2,`$4n_t - 2$`)`
  `ncell(1,2,`$4n_t + 2$`)`
  `xxl(2,2,`$4n_t + 2$`)`
  `nbc(1,1,`$4n_t + 2$`)`, `nbc(1,1,`$4n_t + 3$`)`, ...
- **user-can:** the remaining principal mesh parameters

Explanations:

- `ncell(1,1,2)` is the number of physical cells along the polar angle θ in the angular sector 1 (blocks 4–1–2); it is also the number of cells along the "horizontal" edge of the central block 1;
- `ncell(1,1,3)` is the number of physical cells along the polar angle θ in the angular sector 2 (blocks 5–1–3); it is also the number of cells along the "vertical" edge of the central block 1;
- `xxl(1,1,2)` is the θ coordinate (in degrees of arc) of the lower-left corner of block 2;
- `xxl(nprts(1,2)+1,1,2)` is the θ coordinate (in degrees of arc) of the lower-right corner of block 2;
- `xxl(1,2,2)` is the r coordinate of the lower-left corner of block 2;

– $1 -$ `fdx(1,1,1)` is the stretch factor along angular sector 1 of the mesh in the central block 1 between rectangular and circular shapes;

– $1 -$ `fdx(1,2,1)` is the stretch factor along angular sector 2 of the mesh in the central block 1 between rectangular and circular shapes.

9.9. `igeom=52`: a multi-tier half-circle mesh with a free-float center

1. General description

Here we construct a half-circle quasi-polar mesh with a free-float center, labeled as an H_2 mesh (a "Hohlraum" mesh extending over 2 quadrants of the polar angle θ). It is suitable for both the Cartesian xy and the cylindrical rz geometries with `iradial` $= 0,\ 1,\ 2$. The *core* of the H_2-mesh consists of $n_t \geq 1$ full radial tiers. Each full tier extends over 180° and comprises three neighboring blocks. With one extra block in the center (which is chosen to be the block # 1), the total number of blocks in the mesh core is $3n_t + 1$.

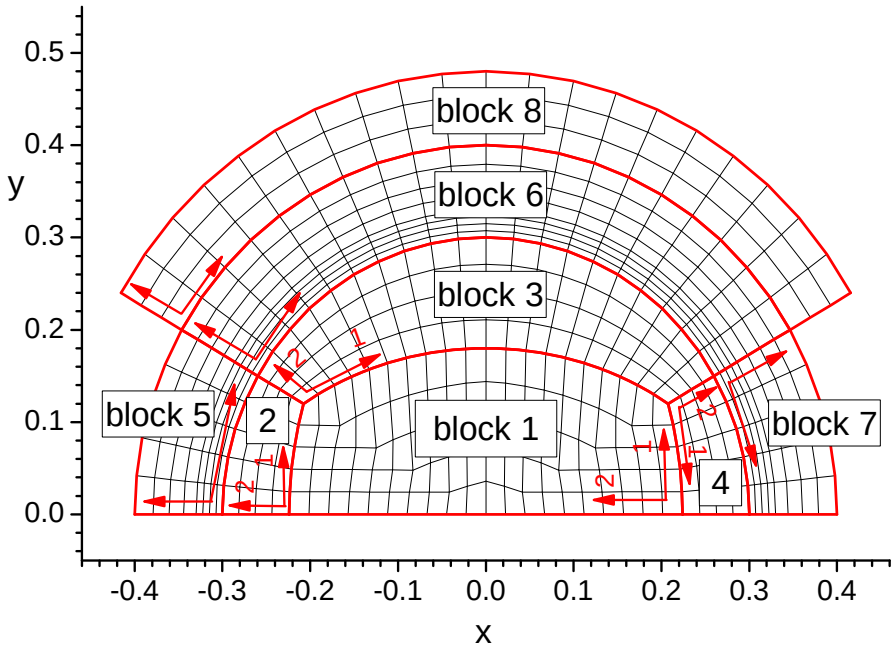


FIG. 9.9: An 8-block half-circle mesh with a free-float center, 7 core blocks, and one "cap" block (block 8).

Beside the $3n_t + 1$ *core blocks*, the H_2-mesh can have 1 or 2 extra protruding blocks (*cap blocks*) in the unfilled $n_t + 1$-th tier, i.e. the total number of blocks in the H_2-mesh is

$$4 \leq 3n_t + 1 \leq \texttt{nblks} \leq 3n_t + 3. \tag{9.25}$$

Generally, different cap blocks must not touch one another along their side edges.

An example of the H_2-mesh with $n_t = 2$ full tiers and one extra cap block is shown in Fig. 9.9. The H_2-mesh is constructed in the progressive-mesh mode only, i.e. all the mesh dimensions are supposed to be set by assigning the arrays `xxl(ns+1,2,nb)` and `fdx(ns,2,nb)`; the values of the mesh parameters `x0(2,nb)` and `dx(ns,2,nb)` are not used.

The full set of parameters that completely specify the physical part of the H_2 mesh consists of the following variables and arrays

$$\begin{aligned}
&\texttt{nblks},\\
&\texttt{nprts(1:2,1:nb)},\\
&\texttt{ncell(1:ns,1:2,1:nb)},\\
&\texttt{xxl(1:ns+1,1:2,1:nb)},\\
&\texttt{fdx(1:ns,1:2,1:nb)},\\
&\texttt{nbc(1:2,1:4,1:nb)}.
\end{aligned} \tag{9.26}$$

The mesh directions $\texttt{m} = 1$ and $\texttt{m} = 2$ are rigidly fixed within every block: in all blocks other than the central block 1 mesh direction 1 is always along the polar θ angle (as measured from the vertical axis in the clockwise direction), while mesh direction 2 is always along the polar radius r.

The mesh in the central block # 1 can be continuously stretched between two extreme configurations: a rectangle and a half-circle. The amount of stretch is independent along the two perpendicular directions and is controlled by the values of the two parameters $0 \le \texttt{fdx(1,1,1)} \le 1$ (the amount of stretch between blocks 4–1–2) and $0 \le \texttt{fdx(1,2,1)} \le 1$ (the amount of stretch between blocks 1–3). For the default values of $\texttt{fdx(1,1,1)} = \texttt{fdx(1,2,1)} = 1$ one obtains a rectangular shape of the central block. If the user sets $\texttt{fdx(1,1,1)} = \texttt{fdx(1,2,1)} = 0$, the result will be a circular central block. The two upper (along the y-axis) corners of block # 1 and the corresponding half-diagonals do not participate in the stretch. The mesh in Fig. 9.9 has been constructed with the values of $\texttt{fdx(1,1,1)} = \texttt{fdx(1,2,1)} = 0.5$.

In practice, to construct the H_2 mesh, one does not need the full set of parameters (9.26) but only its certain subset. We call the subset of the full list (9.26) that is actually used for mesh construction the *principal* mesh parameters. As usual, some of the principal mesh parameters have default values and must not (but can) be specified by the user, others do not have default values and must be specified in the `namelist/input/`.

A special feature of the H_2 mesh is that every individual block can be divided into parts in an arbitrary manner, independently of how other blocks are divided into parts. In other words, the values of `nprts(m,iblk)` can be chosen independently for each mesh direction $\texttt{m} = 1, 2$ in every block $\texttt{iblk} = 1, 2, \ldots, \texttt{nblks}$. But then, because contacting blocks must have the same number of cells along common edges, the user-defined values of `ncell(iprt,m,iblk)` must satisfy certain consistency conditions.

2. Principal mesh parameters

igeom=52: mesh parameters for the θ-direction

Parameters used for the physical θ-mesh construction:

$$\left.\begin{aligned}
&\texttt{nprts(1,iblk)},\\
&\texttt{ncell(1:nprts(1,iblk),1,iblk)},\\
&\texttt{xxl(1:nprts(1,iblk)+1,1,iblk)},\\
&\texttt{fdx(1:nprts(1,iblk),1,iblk)},
\end{aligned}\right\}\ \texttt{iblk} = 2, 3, 4. \tag{9.27}$$

Parameters that must be specified in the `namelist/input/`:

$$\begin{array}{l} \texttt{ncell(1,1,2)}, \\ \texttt{ncell(1,1,3)}, \\ \texttt{xxl(1,1,2)}, \\ \texttt{xxl(2,1,2)}. \end{array} \tag{9.28}$$

Consistency conditions:

$$\sum_j \texttt{ncell(j,1,4)} = \sum_i \texttt{ncell(i,1,2)}; \tag{9.29}$$

$$\begin{array}{l} \texttt{xxl(1,1,iblk+1)} = \texttt{xxl(nprts(1,iblk)+1,1,iblk)}, \quad \texttt{iblk} = 2,3, \\ \texttt{xxl(nprts(1,4)+1,1,4)} = \texttt{xxl(1,1,2)} + 180, \\ \texttt{xxl(1,1,3)} + \texttt{xxl(nprts(1,3)+1,1,3)} = 2 \cdot \texttt{xxl(1,1,iblk)} + 180. \end{array} \tag{9.30}$$

Note that the values of `xxl(i,1,iblk)` for `iblk` = 2–4 specify the angular boundaries of the corresponding block parts in degrees of arc (as measured from the vertical axis in the clockwise direction). If only the values of `xxl(1:2,1,2)` are assigned in the `namelist/input/`, then the values of `xxl(1:2,1,iblk)` for `iblk` = 3 and 4 are calculated from the consistency conditions (9.30).

`igeom=52`: mesh parameters for the r-direction in the core blocks

Parameters used for the physical r-mesh construction:

$$\begin{array}{l} \left.\begin{array}{l} \texttt{nprts(2,iblk)}, \\ \texttt{ncell(1:nprts(2,iblk),2,iblk)}, \end{array}\right\} \texttt{iblk} = 2 \to 3n_t + 1. \\ \left.\begin{array}{l} \texttt{xxl(1:nprts(2,iblk)+1,2,iblk)}, \\ \texttt{fdx(1:nprts(2,iblk),2,iblk)}, \end{array}\right\} \texttt{iblk} = 2, 5, \ldots, 3n_t - 1. \end{array} \tag{9.31}$$

Parameters that must be specified in the `namelist/input/`:

$$\begin{array}{l} \left.\begin{array}{l} \texttt{ncell(1,2,iblk)}, \\ \texttt{xxl(1,2,iblk)}, \end{array}\right\} \texttt{iblk} = 2, 5, \ldots, 3n_t - 1. \\ \texttt{xxl(2,2,}3n_t - 1\texttt{)}. \end{array} \tag{9.32}$$

Consistency conditions:

$$\texttt{xxl(1,2,iblk+3)} = \texttt{xxl(nprts(2,iblk)+1,2,iblk)}, \quad \texttt{iblk} = 2, 3, \ldots, 3n_t - 2. \tag{9.33}$$

$$\texttt{xxl(i,2,iblk)} < \texttt{xxl(i+1,2,iblk)}, \quad \texttt{iblk} = 2, 3, \ldots, 3n_t + 1. \tag{9.34}$$

`igeom=52`: mesh parameters for the "cap" blocks

Parameters used for the physical θ-mesh construction:

$$\left.\begin{array}{l}\texttt{nprts(1,iblk)},\\ \texttt{ncell(1:nprts(1,iblk),1,iblk)},\\ \texttt{nbc(1,1,iblk)},\end{array}\right\}\ \texttt{iblk} = 3n_t + 2 \to \texttt{nblks}. \tag{9.35}$$

Parameters used for the physical r-mesh construction:

$$\left.\begin{array}{l}\texttt{nprts(2,iblk)},\\ \texttt{ncell(1:nprts(2,iblk),2,iblk)},\end{array}\right\}\ \texttt{iblk} = 3n_t + 2 \to \texttt{nblks};$$

$$\left.\begin{array}{l}\texttt{xxl(1:nprts(2,iblk)+1,2,iblk)},\\ \texttt{fdx(1:nprts(2,iblk),2,iblk)},\end{array}\right\}\ \texttt{iblk} = 3n_t + 2; \tag{9.36}$$

Parameters that must be specified in the `namelist/input/`:

$$\left.\begin{array}{l}\texttt{ncell(1,2,iblk)},\\ \texttt{xxl(2,2,iblk)},\end{array}\right\}\ \texttt{iblk} = 3n_t + 2;$$
$$\texttt{nbc(1,1,iblk)}, \quad \texttt{iblk} = 3n_t + 2, \ldots, \texttt{nblks}. \tag{9.37}$$

Consistency conditions:

$$\sum_j \texttt{ncell(j,1,iblk)} = \sum_i \texttt{ncell(i,1,jblk)},$$
$$\texttt{jblk} = \texttt{nbc(1,1,iblk)}, \quad \texttt{iblk} = 3n_t + 2 \to \texttt{nblks}; \tag{9.38}$$

$$\texttt{xxl(1,2,iblk+3)} = \texttt{xxl(nprts(2,iblk)+1,2,iblk)}, \quad \texttt{iblk} = 3n_t - 1 \to \texttt{nblks} - 3. \tag{9.39}$$

$$\texttt{xxl(i,2,iblk)} < \texttt{xxl(i+1,2,iblk)}, \quad \texttt{iblk} = 3n_t + 2 \to \texttt{nblks}. \tag{9.40}$$

`igeom=52`: stretch parameters for the central block # 1

Parameters used to stretch the physical mesh in block # 1:

$$0 \le \texttt{fdx(1,1,1)} \le 1,$$
$$0 \le \texttt{fdx(1,2,1)} \le 1. \tag{9.41}$$

3. Concluding remarks

Having specified the values of the above listed principal mesh parameters, one fully determines the physical part of the H_2 mesh. As usual, the relative width of the ghost-cell layer along the outer boundary is controlled by the user-defined parameter `ghwidth` (default value 10^{-4}). The type of the outer boundary condition is specified by the user-defined values of `ibc(ib,iblk)` along the outer block edges that border vacuum.

However, there remains freedom for dividing into parts those mesh blocks, whose values of `nprts(m,iblk)` and `ncell(ip,m,iblk)` are not required for mesh construction. Hence,

the user can, in addition, specify the values of these parameters in all the remaining blocks as well — provided that they are consistent with the number of cells along the common edges with other blocks. Such an additional division into parts can be used for complex distribution of different materials among different parts of any block in the mesh. The latter is facilitated by the fact that the mesh construction routine `MSHB52` does not change the values of `nprts(m,iblk)` and `ncell(ip,m,iblk)`. The mutual consistency of the values of these arrays is checked in the `subroutine MSHP52`, and the job is aborted if an inconsistency is found.

Finally, the assignment rules for the H_2-mesh parameters can be summarized as follows.

- **fixed or restricted:** `nblks` ≥ 4,
  `ibc(ib,iblk)`$=5$ at common block faces.
- **user-must:** `nblks`
  `ncell(1,1,2)`
  `ncell(1,1,3)`
  `xxl(1:2,1,2)`
  `ncell(1,2,2)`, `ncell(1,2,5)`, ...
  `xxl(1,2,2)`, `xxl(1,2,5)`, ...
  `xxl(2,2,`$3n_t - 1$`)`
  `ncell(1,2,`$3n_t + 2$`)`
  `xxl(2,2,`$3n_t + 2$`)`
  `nbc(1,1,`$3n_t + 2$`)`, `nbc(1,1,`$3n_t + 3$`)`, ...
- **user-can:** the remaining principal mesh parameters

Explanations:

- `ncell(1,1,2)` is the number of physical cells along the polar angle θ in the angular sector 1 (blocks 4–1–2); it is also the number of cells along the "horizontal" edge of the central block 1;
- `ncell(1,1,3)` is the number of physical cells along the polar angle θ in the angular sector 2 (blocks 1–3); it is also the number of cells along the "vertical" edge of the central block 1;
- `xxl(1,1,2)` is the θ coordinate (in degrees of arc) of the lower-left corner of block 2;
- `xxl(nprts(1,2)+1,1,2)` is the θ coordinate (in degrees of arc) of the lower-right corner of block 2;
- `xxl(1,2,2)` is the r coordinate of the lower-left corner of block 2;
- $1 -$ `fdx(1,1,1)` is the stretch factor along angular sector 1 of the mesh in the central block 1 between rectangular and circular shapes;
- $1 -$ `fdx(1,2,1)` is the stretch factor along angular sector 2 of the mesh in the central block 1 between rectangular and circular shapes.

9.10. `igeom=211`: a multi-block randomized x-y or r-z mesh

This is a distorted version of mesh constructed with `IGEOM` = 1–4. The distortion is set by shifting each inner vertex of each block to a random position on a circle of radius $0.2h$ around the original vertex location, where h is a minimum of the four distances to the four principal neighbors of the vertex in question; see Fig. 9.10. The vertices along the block edges are left at their original positions. Such a "random" mesh was proposed in Ref. [**?**].

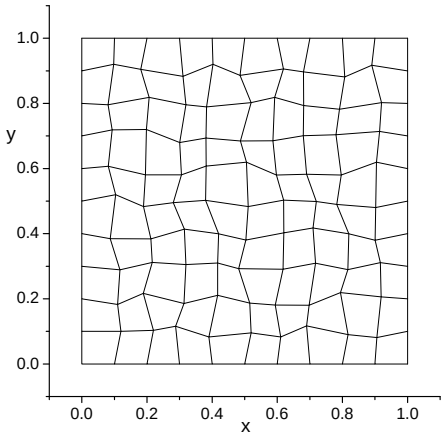


FIG. 9.10: A "random" 20%-distorted mesh in a single block $(x, y) \in [0, 1] \times [0, 1]$.

Mesh parameters for `IGEOM` = 211 are the same as for `IGEOM` = 1, 2, 3 or 4.

9.11. `igeom=212`: a multi-block smoothly distorted x-y or r-z mesh

This is a distorted version of mesh constructed with `IGEOM` = 1, 2. In each block the distortion is set by a sine-modulated diagonal shift for each inner vertex of the originally generated mesh $\{x_{ij}, y_{ij}\}$

$$\begin{aligned} x'_{ij} &= x_{ij} + a_0 \, \sin(2\pi\xi_{ij}) \, \sin(2\pi\eta_{ij}) \,, \\ y'_{ij} &= y_{ij} + a_0 \, \sin(2\pi\xi_{ij}) \, \sin(2\pi\eta_{ij}) \,, \end{aligned} \tag{9.42}$$

where

$$\xi_{ij} \;=\; \frac{\left[(x_{ij} - x_{1j})^2 + (y_{ij} - y_{1j})^2\right]^{1/2}}{\left[(x_{N_x+1,j} - x_{1j})^2 + (y_{N_x+1,j} - y_{1j})^2\right]^{1/2}}, \tag{9.43}$$

$$\eta_{ij} \;=\; \frac{\left[(x_{ij} - x_{i1})^2 + (y_{ij} - y_{i1})^2\right]^{1/2}}{\left[(x_{i,N_y+1} - x_{i1})^2 + (y_{i,N_y+1} - y_{i1})^2\right]^{1/2}}; \tag{9.44}$$

see Fig. 9.11. The vertices along the block edges are left at their original positions. The parent mesh $\{x_{ij}, y_{ij}\}$ must have straight coordinate lines in the (x, y) plane. Evidently, transformation (9.42)–(9.44) can also be applied to the mesh option `IGEOM` = 22.

Mesh parameters for `IGEOM` = 212 are the same as for `IGEOM` = 1 or 2.

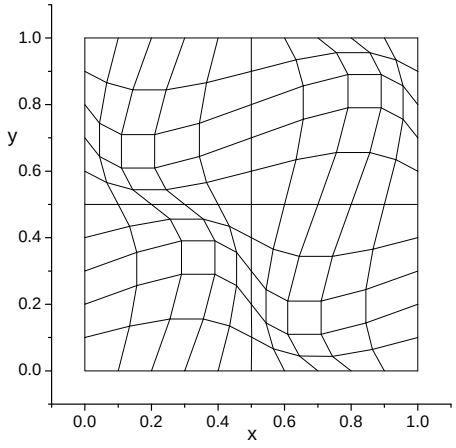


FIG. 9.11: A "wavy" distorted mesh in a single block $(x, y) \in [0, 1] \times [0, 1]$ with the perturbation amplitude $a_0 = 0.1$.

9.12. `igeom=311`: a multi-block randomized polar r-ϕ mesh in cylindrical (x, y) geometry

This is a distorted version of mesh constructed with `IGEOM` = 3. The distortion is set by shifting each inner vertex of each block to a random position on a circle of radius $0.2h$ around the original vertex location, where h is a minimum of the four distances to the four principal neighbors of the vertex in question; see Fig. 9.12. The vertices along the block edges are left at their original positions.

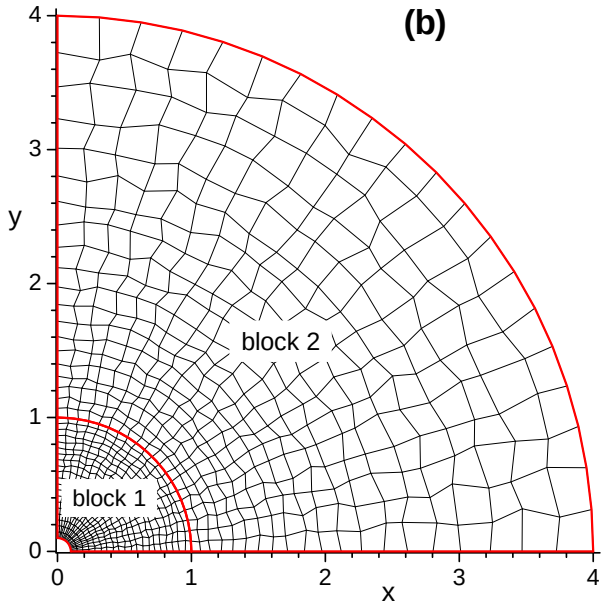


FIG. 9.12: A 2-block "random" 20%-distorted mesh in a quarter-circle.

Mesh parameters for `IGEOM` = 311 are the same as for `IGEOM` = 3.

9.13. `igeom=513`: a multi-tier full-circle mesh of polygon-mosaic type in the r, θ-coordinates

1. *General description*

Topologically, this mesh is equivalent to the above described H_4 mesh for `igeom` = 51. The only difference is that now in blocks 2, 3, ..., `nblks` the mesh geometry in the r, θ-coordinates is piece-wise linear with a polygon-mosaic part-by-part structure — exactly as for `igeom` = 23 in the x, y-coordinates. We will use the name H_4-mosaic for this type of mesh. This mesh can be constructed for `iradial` = 0 only.

So far, construction of the *cap* blocks has not been implemented, and the mesh consists of $n_t \geq 1$ full radial tiers representing the *core* of the mesh. Each full tier consists of 4 neighboring blocks — but now only the *primary circle* of the mesh is assumed to span exactly 180°. The primary mesh circle is defined as a full circle composed of the three lower (`ib` = 1) edges of blocks 2, 3, 4, and 5 before the stretch transformation. With one extra block in the center (which is chosen to be the block # 1), the total number of blocks in the mesh core is $4n_t + 1$.

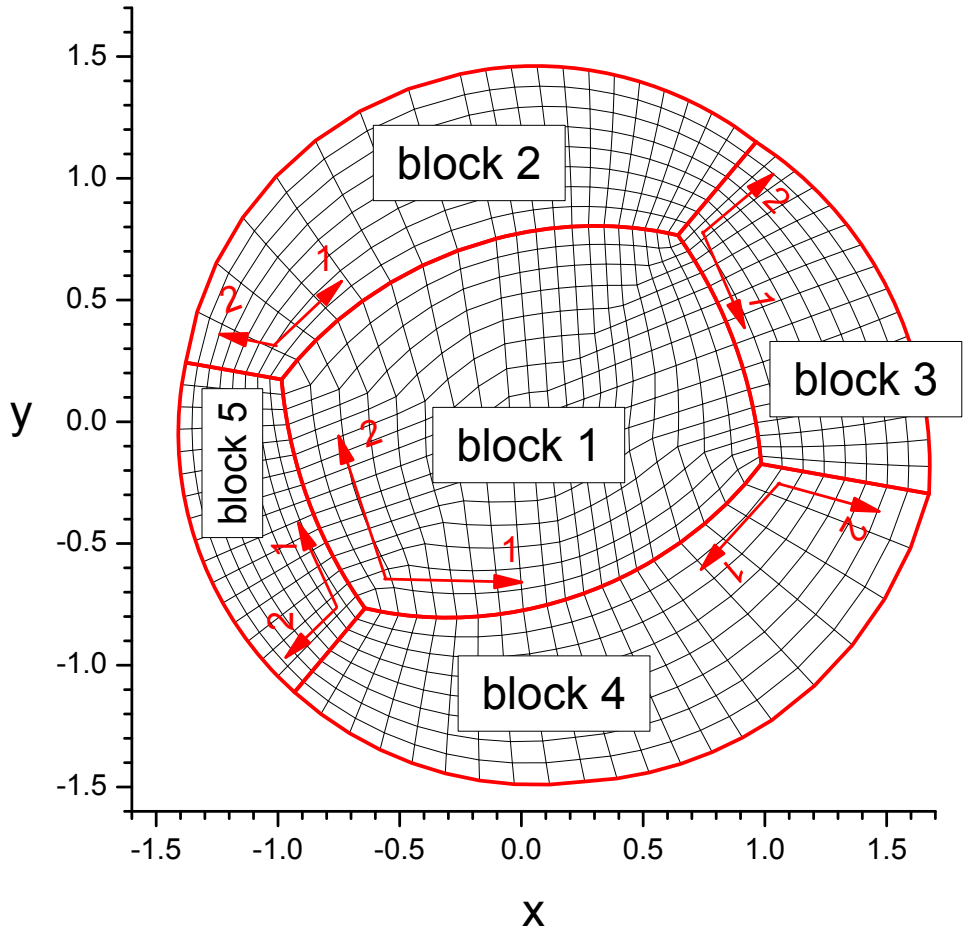


FIG. 9.13: A 5-block full-circle H_4-mosaic mesh with a polygon-mosaic (r, θ) structure.

Beside the $4n_t + 1$ *core blocks*, this mesh can have up to 3 extra protruding blocks (*cap*

blocks) in the unfilled $n_t + 1$-th tier, i.e. the total number of blocks is

$$4 \le 4n_t + 1 \le \texttt{nblks} \le 4n_t + 4. \tag{9.45}$$

Generally, different cap blocks must not touch one another along their side edges. An example of the `igeom` = 513 mesh with a single full tier is shown in Fig. 9.13.

The full set of parameters that completely specify the physical part of the H_4-mosaic mesh includes the following variables and arrays

```
nblks,
nprts(1:2,1:nb),
ncell(1:ns,1:2,1:nb),
xxp(1:ns+1,1:ns+1,1:2,2:nb),
fdxp(1:ns+1,1:ns+1,1:2,1:nb),
nbc(1:2,1:4,1:nb).
```
(9.46)

The mesh directions `m` = 1 and `m` = 2 are rigidly fixed within every block: in all blocks, other than the central block 1, mesh direction 1 is always along the polar θ angle (as measured from the vertical axis in the clockwise direction), while mesh direction 2 is always along the polar radius r. Similarly to the case of `igeom` = 23, the mesh parameters `xxp` and `fdxp` specify the corner coordinates and the cell-size progression ratios for individual parts within every block, starting from block #2.

The mesh in the central block #1 can be continuously stretched between two extreme configurations: a rectangle and a full circle. The amount of stretch is independent along the two perpendicular directions and is controlled by the values of the two parameters $0 \le$ `fdxp(1,1,1,1)` ≤ 1 (the amount of stretch between blocks 5–1–3) and $0 \le$ `fdxp(1,1,2,1)` ≤ 1 (the amount of stretch between blocks 4–1–2). For the default values of `fdxp(1,1,1,1)` = `fdxp(1,1,2,1)` = 1 one obtains a rectangular shape of the central block. If the user sets `fdxp(1,1,1,1)` = `fdxp(1,1,2,1)` = 0, the result will be a circular central block. The four corners of block #1 and its diagonals do not participate in the stretch. The mesh in Fig. 9.13 was constructed with the values of `fdxp(1,1,1,1)` = `fdxp(1,1,2,1)` = 0.5.

In practice, to construct the H_4-mosaic mesh, one does not need the full set of parameters (9.46) but only its certain subset. We call the subset of the full list (9.46), which is actually used for mesh construction, the *principal* mesh parameters. As usual, some of the principal mesh parameters have default values and must not (but can) be specified by the user, others do not have default values and must be specified in the `namelist/input/`.

A special feature of the H_4-mosaic mesh is that every individual block can be divided into parts in an arbitrary manner, independently of the part structure of other blocks. In other words, the values of `nprts(m,iblk)` can be chosen independently for each mesh direction `m` = 1, 2 in every block `iblk` = $1, 2, \ldots,$ `nblks`. But then, because contacting blocks must have the same number of cells along common edges, the user-defined values of `ncell(iprt,m,iblk)` must satisfy certain consistency conditions.

2. Principal mesh parameters

`igeom=513`: mesh parameters for the θ-direction

Parameters used for the physical θ-mesh construction:

$$\left.\begin{array}{l}\texttt{nprts(1,iblk)},\\ \texttt{ncell(1:nprts(1,iblk),1,iblk)},\\ \texttt{xxp(1:nprts(1,iblk)+1,1:nprts(2,iblk)+1,1,iblk)},\\ \texttt{fdxp(1:nprts(1,iblk),1:nprts(2,iblk)+1,1,iblk)},\end{array}\right\}\ \texttt{iblk} = 2,3,\ldots,\texttt{nblks}. \tag{9.47}$$

Parameters that must be specified in the `namelist/input/`:

$$\begin{array}{l}\texttt{ncell(1,1,2)},\\ \texttt{ncell(1,1,3)},\\ \texttt{xxp(1:2,1:2,1,2:nblks)},\end{array} \tag{9.48}$$

except for `xxp(1:2,1,1,3)`, `xxp(1:2,1,1,4)`, and `xxp(1:2,1,1,5)`.

Consistency conditions:

$$\begin{array}{l}\displaystyle\sum_j \texttt{ncell(j,1,4)} = \sum_i \texttt{ncell(i,1,2)},\\ \displaystyle\sum_j \texttt{ncell(j,1,5)} = \sum_i \texttt{ncell(i,1,3)},\end{array} \tag{9.49}$$

$$\begin{array}{l}\texttt{xxp(1,1,1,iblk+1)} = \texttt{xxp(nprts(1,iblk)+1,1,1,iblk)}, \quad \texttt{iblk} = 2,3,4,\\ \texttt{xxp(1,1,1,4)} = \texttt{xxp(1,1,1,2)} + 180,\\ \texttt{xxp(1,1,1,5)} = \texttt{xxp(1,1,1,3)} + 180.\end{array} \tag{9.50}$$

Also, one should of course pay attention that the coordinates of all meeting corners of different blocks have the same values.

Note that the values of `xxp(ip,jp,1,iblk)` for `iblk` = 2–`nblks` specify the angular boundaries of the corresponding block parts in degrees of arc (as measured from the vertical y-axis in the clockwise direction). If just the values of `xxp(1:2,1,1,2)` are assigned in the `namelist/input/` [for `nprts(1,2)` = 1], then the values of `xxp(1:2,1,1,iblk)` for `iblk` = 3–5 are calculated from the consistency conditions (9.50).

igeom=513: mesh parameters for the r-direction in the core blocks

Parameters used for the physical r-mesh construction:

$$\left.\begin{array}{l}\texttt{nprts(2,iblk)},\\ \texttt{ncell(1:nprts(2,iblk),2,iblk)},\end{array}\right\}\ \texttt{iblk} = 2,3,\ldots,\texttt{nblks}.$$

$$\left.\begin{array}{l}\texttt{xxp(1:nprts(1,iblk)+1,1:nprts(2,iblk)+1,2,iblk)},\\ \texttt{fdxp(1:nprts(1,iblk)+1,1:nprts(2,iblk),2,iblk)},\end{array}\right\}\ \texttt{iblk} = 2,3,\ldots,\texttt{nblks}. \tag{9.51}$$

Parameters that must be specified in the `namelist/input/`:

$$\begin{array}{l}\texttt{ncell(1,2,iblk)}, \quad \texttt{iblk} = 2,6,\ldots,4n_t-2.\\ \texttt{xxp(1:2,1:2,2,iblk)}, \quad \texttt{iblk} = 2,3,\ldots,\texttt{nblks},\end{array} \tag{9.52}$$

except for `xxp(2,1,2,2)`, `xxp(1:2,1,2,3)`, `xxp(1:2,1,2,4)`, and `xxp(1:2,1,2,5)` because

$$\texttt{xxp(2,1,2,2)} = \texttt{xxp(1:2,1,2,3)} = \texttt{xxp(1:2,1,2,4)} = \texttt{xxp(1:2,1,2,5)} = \texttt{xxp(1,1,2,2)} \tag{9.53}$$

is the radius of the primary circle fixed by the specified value of `xxp(1,1,2,2)`.

igeom=513: mesh parameters for the "cap" blocks

Parameters used for the physical θ-mesh construction:

$$\left.\begin{array}{l}\texttt{nprts(1,iblk)},\\ \texttt{ncell(1:nprts(1,iblk),1,iblk)},\\ \texttt{nbc(1,1,iblk)},\\ \texttt{xxp(1:nprts(1,iblk)+1,1:nprts(2,iblk)+1,1,iblk)},\\ \texttt{fdxp(1:nprts(1,iblk),1:nprts(2,iblk)+1,1,iblk)},\end{array}\right\}\ \texttt{iblk} = 4n_t + 2 \rightarrow \texttt{nblks}. \tag{9.54}$$

Parameters used for the physical r-mesh construction:

$$\left.\begin{array}{l}\texttt{nprts(2,iblk)},\\ \texttt{ncell(1:nprts(2,iblk),2,iblk)},\\ \texttt{xxp(1:nprts(1,iblk)+1,1:nprts(2,iblk)+1,2,iblk)},\\ \texttt{fdxp(1:nprts(1,iblk)+1,1:nprts(2,iblk),2,iblk)},\end{array}\right\}\ \texttt{iblk} = 4n_t + 2 \rightarrow \texttt{nblks}. \tag{9.55}$$

igeom=513: stretch parameters for the central block # 1

Parameters used to stretch the physical mesh in block #1:

$$\begin{array}{l}0 \leq \texttt{fdx(1,1,1,1)} \leq 1,\\ 0 \leq \texttt{fdx(1,1,2,1)} \leq 1.\end{array} \tag{9.56}$$

Example:

The mesh shown in Fig. 9.13 has been constructed with the following set of parameters:

- geometry:
  ```
  igeom=513
  iradial=0
  nblks=5
  ```

- theta-mesh:
  ```
  nprts(1,2)=2
  nprts(1,3)=2
  nprts(1,4)=3
  ncell(1,1,2)=8,12
  ncell(1,1,3)=8,8
  ncell(1,1,4)=5,6,9
  ncell(1,1,5)=16
  ```

```
xxp(1,1,1,2)=-80.d0, -10.d0, 40.d0
xxp(1,2,1,2)=-80.d0, -10.d0, 40.d0
fdxp(1,1,1,2)=1.05d0, .95d0
xxp(1,1,1,3)=40.d0, 70.d0, 100.d0
xxp(1,2,1,3)=40.d0, 70.d0, 100.d0
fdxp(1,1,1,3)=1.15d0, .85d0
xxp(1,1,1,4)=100.d0, 140.d0, 170.d0, 220.d0
xxp(1,2,1,4)=100.d0, 140.d0, 170.d0, 220.d0
fdxp(1,1,1,4)=1.03d0, 1.00d0, .97d0
xxp(1,1,1,5)=220.d0, 280.d0
xxp(1,2,1,5)=220.d0, 280.d0
```

- block 1 is stretched 50%:

```
fdxp(1,1,1,1)=0.5d0
fdxp(1,1,2,1)=0.5d0
```

- r-mesh:

```
ncell(1,2,2)=8
ncell(1,2,3)=8
ncell(1,2,4)=8
ncell(1,2,5)=8
xxp(1,1,2,2)=1.d0, 1.0d0, 1.d0
xxp(1,2,2,2)=1.4d0, 1.45d0, 1.5d0
xxp(1,1,2,3)=1.d0, 1.d0, 1.d0
xxp(1,2,2,3)=1.5d0, 1.6d0, 1.7d0
xxp(1,1,2,4)=1.d0, 1.d0, 1.d0, 1.d0
xxp(1,2,2,4)=1.7d0, 1.6d0, 1.5d0, 1.45d0
xxp(1,1,2,5)=1.d0, 1.d0
xxp(1,2,2,5)=1.45d0, 1.4d0
```

3. Concluding remarks

Having specified the values of the above listed principal mesh parameters, one fully determines the physical part of the H_4-mosaic mesh. As usual, the relative width of the ghost-cell layer along the outer boundary is controlled by the user-defined parameter `ghwidth` (default value 10^{-4}). The type of the outer boundary condition is specified by the user-defined values of `ibc(ib,iblk)` along the outer block edges that border vacuum. Mutual consistency of the mesh parameters assigned by the user is partially checked in the `subroutine MSHP513`, and the job is aborted if any inconsistency is found.

Explanations:

- `nprts(m,iblk)` is the number of parts along mesh direction `m` in block `iblk`;
- `ncell(iprt,m,iblk)` is the number of physical cells along mesh direction `m` in part `iprt` (along this direction) of block `iblk`;
- `xxp(iprt,jprt,1,iblk)` is the θ $(= x_1)$ coordinate (in degrees of arc as measured from the vertical x_2-axis) of the lower-left corner of part `(iprt,jprt)` in block `iblk`; `iprt` is the part index along mesh direction 1, `jprt` is the part index along mesh direction 2; correspondingly, the lower-right, upper-left, and upper-right corners of part `(iprt,jprt)` in block `iblk` have the x-coordinates `xxp(iprt+1,jprt,1,iblk)`,

xxp(iprt,jprt+1,1,iblk), and xxp(iprt+1,jprt+1,1,iblk); here iblk = $2, 3, \ldots,$ nblks; default value is undef;

- xxp(iprt,jprt,2,iblk) is the r ($= x_2$) coordinate of the lower-left corner of part (iprt,jprt) in block iblk; here iblk $= 2, 3, \ldots,$ nblks; default value is undef;
- fdxp(iprt,jprt,1,iblk) is the common ratio of successive cell sizes along the part edge, starting from the lower-left corner of part (iprt,jprt) along mesh direction 1, in block iblk; correspondingly, fdxp(iprt,jprt+1,1,iblk) is the common ratio of successive cell sizes along the part edge, starting from the upper-left corner of part (iprt,jprt) along mesh direction 1; here iblk $= 2, 3, \ldots,$ nblks; default value is 1.0;
- fdxp(iprt,jprt,2,iblk) is the common ratio of successive cell sizes along the part edge, starting from the lower-left corner of part (iprt,jprt) along mesh direction 2, in block iblk; correspondingly, fdxp(iprt+1,jprt,1,iblk) is the common ratio of successive cell sizes along the part edge, starting from the lower-right corner of part (iprt,jprt) along mesh direction 2; here iblk $= 2, 3, \ldots,$ nblks; default value is 1.0;
- xxp(1,1,2,2) is the radius of the primary circle of the H_4-mosaic mesh;
- $1 -$ fdxp(1,1,1,1) is the stretch factor along angular sector 1 of the mesh in the central block 1 between rectangular and circular shapes;
- $1 -$ fdxp(1,1,2,1) is the stretch factor along angular sector 2 of the mesh in the central block 1 between rectangular and circular shapes.

9.14. igeom=523: a multi-tier half-circle mesh of polygon-mosaic type in the r, θ-coordinates

1. General description

Topologically, this mesh is equivalent to the above described H_2 mesh for igeom = 52. The only difference is that now in blocks 2, 3, ..., $3n_t + 1$ the mesh geometry in the r, θ-coordinates is piece-wise linear with a polygon-mosaic part-by-part structure — exactly as for igeom = 23 in the x, y-coordinates. We will use the name H_2-mosaic for this type of mesh.

So far, construction of the *cap* blocks has not been implemented, and the mesh consists of $n_t \geq 1$ full radial tiers representing the *core* of the mesh. Each full tier consists of 3 neighboring blocks — but now only the *primary circle* of the mesh is assumed to span exactly 180°. The primary mesh circle is defined as a circular arc composed of the three lower (ib = 1) edges of blocks 2, 3, and 4 before the stretch transformation. With one extra block in the center (which is chosen to be the block # 1), the total number of blocks in the mesh core is $3n_t + 1$.

Beside the $3n_t + 1$ *core blocks*, this mesh can have 1 or 2 extra protruding blocks (*cap blocks*) in the unfilled $n_t + 1$-th tier, i.e. the total number of blocks is

$$4 \leq 3n_t + 1 \leq \texttt{nblks} \leq 3n_t + 3. \tag{9.57}$$

Generally, different cap blocks must not touch one another along their side edges. An example of the igeom = 523 mesh with a single full tier is shown in Fig. 9.14.

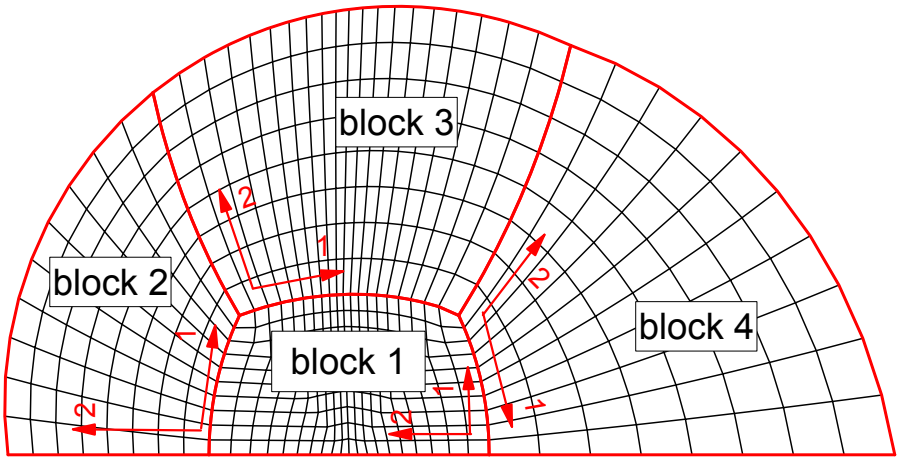


FIG. 9.14: A 4-block half-circle H_2-mosaic mesh with a polygon-mosaic (r, θ) structure.

The full set of parameters that completely specify the physical part of the H_2-mosaic mesh includes the following variables and arrays

```
nblks,
nprts(1:2,1:nb),
ncell(1:ns,1:2,1:nb),
xxp(1:ns+1,1:ns+1,1:2,2:nb),
fdxp(1:ns+1,1:ns+1,1:2,1:nb),
nbc(1:2,1:4,1:nb).
```
(9.58)

The mesh directions `m` = 1 and `m` = 2 are rigidly fixed within every block: in all blocks other than the central block 1 mesh direction 1 is always along the polar θ angle (as measured from the vertical axis in the clockwise direction), while mesh direction 2 is always along the polar radius r. Similarly to the case of `igeom` = 23, the mesh parameters `xxp` and `fdxp` specify the corner coordinates and the cell-size progression ratios for individual parts within every block, starting from block #2.

The mesh in the central block #1 can be continuously stretched between two extreme configurations: a rectangle and a half-circle. The amount of stretch is independent along the two perpendicular directions and is controlled by the values of the two parameters $0 \le$ `fdxp(1,1,1,1)` ≤ 1 (the amount of stretch between blocks 4–1–2) and $0 \le$ `fdxp(1,1,2,1)` ≤ 1 (the amount of stretch between blocks 1–3). For the default values of `fdxp(1,1,1,1)` = `fdxp(1,1,2,1)` = 1 one obtains a rectangular shape of the central block. If the user sets `fdxp(1,1,1,1)` = `fdxp(1,1,2,1)` = 0, the result will be a circular central block. The two upper (along the y-axis) corners of block #1 and the corresponding half-diagonals do not participate in the stretch. The mesh in Fig. 9.14 was constructed with the values of `fdxp(1,1,1,1)` = `fdxp(1,1,2,1)` = 0.5.

In practice, to construct the H_2-mosaic mesh, one does not need the full set of parameters (9.58) but only its certain subset. We call the subset of the full list (9.58) that is actually used for mesh construction the *principal* mesh parameters. As usual, some of the principal mesh parameters have default values and must not (but can) be specified by the user, others do not have default values and must be specified in the `namelist/input/`.

A special feature of the H_2-mosaic mesh is that every individual block can be divided into parts in an arbitrary manner, independently of the part structure of other blocks.

In other words, the values of `nprts(m,iblk)` can be chosen independently for each mesh direction `m` $= 1, 2$ in every block `iblk` $= 1, 2, \ldots,$ `nblks`. But then, because contacting blocks must have the same number of cells along common edges, the user-defined values of `ncell(iprt,m,iblk)` must satisfy certain consistency conditions.

2. Principal mesh parameters

igeom=523: mesh parameters for the θ-direction

Parameters used for the physical θ-mesh construction:

$$\left.\begin{array}{l}\texttt{nprts(1,iblk)},\\ \texttt{ncell(1:nprts(1,iblk),1,iblk)},\\ \texttt{xxp(1:nprts(1,iblk)+1,1:nprts(2,iblk)+1,1,iblk)},\\ \texttt{fdxp(1:nprts(1,iblk),1:nprts(2,iblk)+1,1,iblk)},\end{array}\right\}\ \texttt{iblk} = 2, 3, \ldots, \texttt{nblks}. \tag{9.59}$$

Parameters that must be specified in the `namelist/input/`:

$$\begin{array}{l}\texttt{ncell(1,1,2)},\\ \texttt{ncell(1,1,3)},\\ \texttt{xxp(1:2,1:2,1,2:nblks)},\end{array} \tag{9.60}$$

except for `xxp(1:2,1,1,3)` and `xxp(1:2,1,1,4)`.

Consistency conditions:

$$\sum_j \texttt{ncell(j,1,4)} = \sum_i \texttt{ncell(i,1,2)}, \tag{9.61}$$

$$\begin{array}{l}\texttt{xxp(1,1,1,iblk+1)} = \texttt{xxp(nprts(1,iblk)+1,1,1,iblk)}, \quad \texttt{iblk} = 2, 3,\\ \texttt{xxp(nprts(1,4)+1,1,1,4)} = \texttt{xxp(1,1,1,2)} + 180,\\ \texttt{xxp(1,1,1,3)} + \texttt{xxp(nprts(1,3)+1,1,1,3)} = 2 \cdot \texttt{xxp(1,1,1,iblk)} + 180.\end{array} \tag{9.62}$$

Also, one should of course pay attention that the coordinates of all meeting corners of different blocks have the same values.

Note that the values of `xxp(ip,jp,1,iblk)` for `iblk` $= 2$–`nblks` specify the angular boundaries of the corresponding block parts in degrees of arc (as measured from the vertical y-axis in the clockwise direction). If only the values of `xxp(1:2,1,1,2)` are assigned in the `namelist/input/` [for `nprts(1,2)` $= 1$], then the values of `xxp(1:2,1,1,iblk)` for `iblk` $= 3$ and 4 are calculated from the consistency conditions (9.62).

igeom=523: mesh parameters for the r-direction in the core blocks

Parameters used for the physical r-mesh construction:

$$\left.\begin{array}{l}\texttt{nprts(2,iblk)},\\ \texttt{ncell(1:nprts(2,iblk),2,iblk)},\end{array}\right\}\ \texttt{iblk} = 2,3,\ldots,\texttt{nblks}.$$

$$\left.\begin{array}{l}\texttt{xxp(1:nprts(1,iblk)+1,1:nprts(2,iblk)+1,2,iblk)},\\ \texttt{fdxp(1:nprts(1,iblk)+1,1:nprts(2,iblk),2,iblk)},\end{array}\right\}\ \texttt{iblk} = 2,3,\ldots,\texttt{nblks}. \tag{9.63}$$

Parameters that must be specified in the `namelist/input/`:

$$\begin{array}{l}\texttt{ncell(1,2,iblk)},\quad \texttt{iblk} = 2,5,\ldots,3n_t-1.\\ \texttt{xxp(1:2,1:2,2,iblk)},\quad \texttt{iblk} = 2,3,\ldots,\texttt{nblks},\end{array} \tag{9.64}$$

except for `xxp(2,1,2,2)`, `xxp(1:2,1,2,3)` and `xxp(1:2,1,2,4)` because

$$\texttt{xxp(2,1,2,2)} = \texttt{xxp(1:2,1,2,3)} = \texttt{xxp(1:2,1,2,4)} = \texttt{xxp(1,1,2,2)} \tag{9.65}$$

is the radius of the primary circle fixed by the specified value of `xxp(1,1,2,2)`.

`igeom=523`: mesh parameters for the "cap" blocks

Parameters used for the physical θ-mesh construction:

$$\left.\begin{array}{l}\texttt{nprts(1,iblk)},\\ \texttt{ncell(1:nprts(1,iblk),1,iblk)},\\ \texttt{nbc(1,1,iblk)},\\ \texttt{xxp(1:nprts(1,iblk)+1,1:nprts(2,iblk)+1,1,iblk)},\\ \texttt{fdxp(1:nprts(1,iblk),1:nprts(2,iblk)+1,1,iblk)},\end{array}\right\}\ \texttt{iblk} = 3n_t+2 \rightarrow \texttt{nblks}. \tag{9.66}$$

Parameters used for the physical r-mesh construction:

$$\left.\begin{array}{l}\texttt{nprts(2,iblk)},\\ \texttt{ncell(1:nprts(2,iblk),2,iblk)},\\ \texttt{xxp(1:nprts(1,iblk)+1,1:nprts(2,iblk)+1,2,iblk)},\\ \texttt{fdxp(1:nprts(1,iblk)+1,1:nprts(2,iblk),2,iblk)},\end{array}\right\}\ \texttt{iblk} = 3n_t+2 \rightarrow \texttt{nblks}. \tag{9.67}$$

Parameters that must be specified in the `namelist/input/`:

$$\begin{array}{l}\texttt{ncell(1,2,iblk)},\quad \texttt{iblk} = 3n_t+2,\\ \texttt{nbc(1,1,iblk)},\quad \texttt{iblk} = 3n_t+2,\ldots,\texttt{nblks},\\ \texttt{xxp(1:nprts(1,iblk)+1,1:nprts(2,iblk)+1,1:2,iblk)},\quad \texttt{iblk} = 3n_t+2,\ldots,\texttt{nblks}.\end{array} \tag{9.68}$$

`igeom=523`: stretch parameters for the central block # 1

Parameters used to stretch the physical mesh in block #1:

$$\begin{array}{l}0 \le \texttt{fdx(1,1,1,1)} \le 1,\\ 0 \le \texttt{fdx(1,1,2,1)} \le 1.\end{array} \tag{9.69}$$

Example:

The mesh shown in Fig. 9.14 has been constructed with the following set of parameters:

- geometry:

```
igeom=523
iradial=0
nblks=4
```

- theta-mesh:

```
nprts(1,3)=2
ncell(1,1,2)=10
ncell(1,1,3)=10,10
ncell(1,1,4)=10
xxp(1,1,1,2)=-90.d0, -40.d0
xxp(1,2,1,2)=-90.d0, -30.d0
xxp(1,1,1,3)=-40.d0, 0.d0, 40.d0
xxp(1,2,1,3)=-30.d0, 0.d0, 30.d0
xxp(1,1,1,4)=40.d0, 90.d0
xxp(1,2,1,4)=30.d0, 90.d0
fdxp(1,1,1,3)=.9d0, 1.1d0
fdxp(1,2,1,3)=.9d0, 1.1d0
```

- block 1 is stretched 50%:

```
fdxp(1,1,1,1)=0.5d0
fdxp(1,1,2,1)=0.5d0
```

- r-mesh:

```
ncell(1,2,2)=8
xxp(1,1,2,2)=1.d0, 1.d0
xxp(1,2,2,2)=2.d0, 2.3d0
xxp(1,1,2,3)=1.d0, 1.d0, 1.d0
xxp(1,2,2,3)=2.3d0, 2.45d0, 2.6d0
xxp(1,1,2,4)=1.d0, 1.d0
xxp(1,2,2,4)=2.6d0, 3.2d0
```

3. Concluding remarks

Having specified the values of the above listed principal mesh parameters, one fully determines the physical part of the H_2-mosaic mesh. As usual, the relative width of the ghost-cell layer along the outer boundary is controlled by the user-defined parameter `ghwidth` (default value 10^{-4}). The type of the outer boundary condition is specified by the user-defined values of `ibc(ib,iblk)` along the outer block edges that border vacuum. Mutual consistency of the mesh parameters assigned by the user is partially checked in the `subroutine MSHP523`, and the job is aborted if an inconsistency is found.

Explanations:

- `nprts(m,iblk)` is the number of parts along mesh direction `m` in block `iblk`;
- `ncell(iprt,m,iblk)` is the number of physical cells along mesh direction `m` in part `iprt` (along this direction) of block `iblk`;

– `xxp(iprt,jprt,1,iblk)` is the θ ($= x_1$) coordinate (in degrees of arc as measured from the vertical x_2-axis) of the lower-left corner of part `(iprt,jprt)` in block `iblk`; `iprt` is the part index along mesh direction 1, `jprt` is the part index along mesh direction 2; correspondingly, the lower-right, upper-left, and upper-right corners of part `(iprt,jprt)` in block `iblk` have the x-coordinates `xxp(iprt+1,jprt,1,iblk)`, `xxp(iprt,jprt+1,1,iblk)`, and `xxp(iprt+1,jprt+1,1,iblk)`; here `iblk` $= 2, 3, \ldots,$ `nblks`; default value is `undef`;

– `xxp(iprt,jprt,2,iblk)` is the r ($= x_2$) coordinate of the lower-left corner of part `(iprt,jprt)` in block `iblk`; here `iblk` $= 2, 3, \ldots,$ `nblks`; default value is `undef`;

– `fdxp(iprt,jprt,1,iblk)` is the common ratio of successive cell sizes along the part edge, starting from the lower-left corner of part `(iprt,jprt)` along mesh direction 1, in block `iblk`; correspondingly, `fdxp(iprt,jprt+1,1,iblk)` is the common ratio of successive cell sizes along the part edge, starting from the upper-left corner of part `(iprt,jprt)` along mesh direction 1; here `iblk` $= 2, 3, \ldots,$ `nblks`; default value is 1.0;

– `fdxp(iprt,jprt,2,iblk)` is the common ratio of successive cell sizes along the part edge, starting from the lower-left corner of part `(iprt,jprt)` along mesh direction 2, in block `iblk`; correspondingly, `fdxp(iprt+1,jprt,1,iblk)` is the common ratio of successive cell sizes along the part edge, starting from the lower-right corner of part `(iprt,jprt)` along mesh direction 2; here `iblk` $= 2, 3, \ldots,$ `nblks`; default value is 1.0;

– `xxp(1,1,2,2)` is the radius of the primary circle of the H_2-mosaic mesh;

– $1 -$ `fdxp(1,1,1,1)` is the stretch factor along angular sector 1 of the mesh in the central block 1 between rectangular and circular shapes;

– $1 -$ `fdxp(1,1,2,1)` is the stretch factor along angular sector 2 of the mesh in the central block 1 between rectangular and circular shapes.

9.15. `igeom=2001`: a 4-block (or 5-block) "daisy-like" mesh in a rectangle

This is a special case of a 4-block skewed mesh in a rectangular region, designed primarily for test problems. The optional 5-th block has a height of $y_1 - y_0$ and is attached from below (see Fig. 9.15).

Mesh parameters for `igeom=2001`:

- **fixed:** `nprts(m,iblk) =1`
  `ncell(1,m,iblk)` for `iblk` ≥ 2
  `ibc(ib,iblk)` [except for `ibc(2,1)`, `ibc(3,1)`, `ibc(2,2)`, and `ibc(1,5)` (or `ibc(1,3)`)]
  `nbc(ib,iblk)`
- **user-must:** `ncell(1,1,1)`
  `ncell(1,2,1)`
  `x0(1,1)` $= y_0$
  `x0(2,1)` $= y_1$

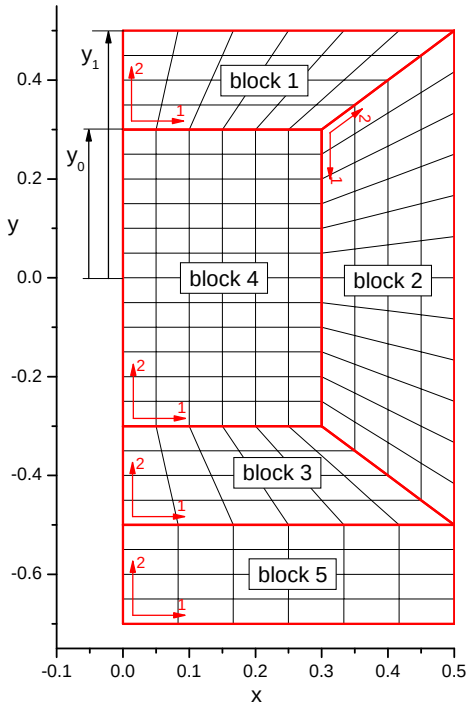


FIG. 9.15: A 5-block "daisy-like" mesh in a rectangular region for `IGEOM` = 2001.

- **user-can:**

`iradial`	def $= 0$
`nblks` ($= 4$ or $= 5$)	def $= 4$
`ibc(2,1)`	def $= 2$
`ibc(3,1)`	def $= 2$
`ibc(2,2)`	def $= 2$
`ibc(1,3)` (or `ibc(1,5)` for 5 blocks)	def $= 2$
`ghwidth`	def $= 10^{-4}$

Explanations:

- `ncell(1,1,1)` is the number of physical cells along the short edge of the central rectangle in block 4; its long edge has `2ncell(1,1,1)` mesh cells;
- `ncell(1,2,1)` is the number of physical cells along the radial direction in blocks 1, 2 and 3;
- `x0(1,1)` is the length of the short edge of block 4 equal to y_0;
- `x0(2,1)` is one half of the total extension of the 4-block mesh along y-axis equal to $y_1 > y_0$;

9.16. `igeom=2002`: a 5-block skewed Kershaw mesh in a rectangle

This is a special case of a skewed mesh in a rectangular region proposed by Kershaw [**?**]. In our case this mesh is constructed by using five blocks stacked vertically (along the y-axis)

on top of one another, each having two parts along the horizontal x-axis; see Fig. 9.16. The mesh is constructed by applying the `MSHB22` subroutine (i.e. the AQ-mesh algorithm) to each block. No material differences between different blocks and parts are foreseen in this version. Concerning the material properties and the initial state, it is sufficient to set the necessary parameters (namely, `rho0(1,1)`, `pr0(1,1)`, `matnum(1,1)`) for the first part of the first block only: the same values are then automatically assigned in all other parts and blocks.

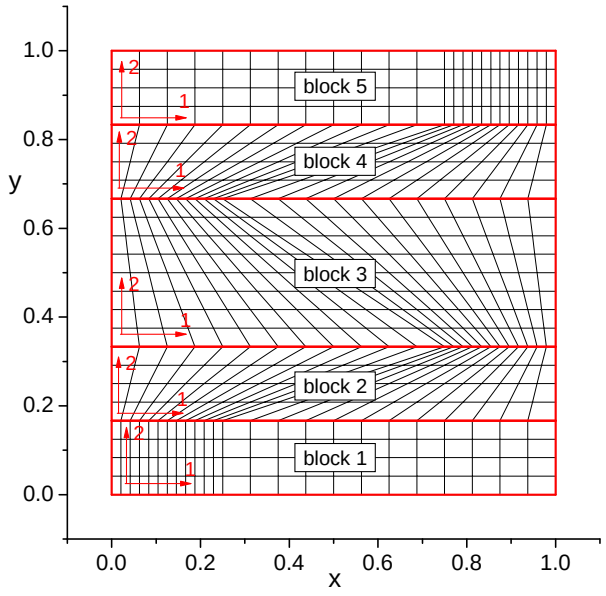


FIG. 9.16: A 5-block skewed Kershaw mesh in a rectangular region for `IGEOM` = 2002.

Identical boundary conditions are assigned along edges 3 and 4 of all five blocks, i.e. `ibc(3:4,iblk)` = `ibc(3:4,1)`, `ITCONBC(3:4,iblk)` = `ITCONBC(3:4,1)`, `TEMPBC(3:4,iblk)` = `TEMPBC(3:4,1)`, ... for all `iblk` = 2, 3, 4, 5.

Mesh parameters for ***igeom=2002:***

- **fixed:** `nblks` = 5
  `nprts(1,iblk)` = 2
  `nprts(2,iblk)` = 1
- **user-must:** `ncell(1,1,1)`
  `ncell(1,2,1)`
  `xxl(1:3,1,1)`
  `xxl(1:2,2,1)`
- **user-can:**

`iradial`	def = 0
`fdxaq(iprt,ib,iblk)`	def = 1.0
`ibc(1,1)`	def = 2
`ibc(3,1)`	def = 2
`ibc(4,1)`	def = 2
`ibc(2,5)`	def = 2
`ghwidth`	def = 10^{-4}

Explanations:

- `ncell(1,1,1)` is the number of physical cells in part 1 of all five blocks along x-axis; part 2 contains equal number of cells;
- `ncell(1,2,1)` is the number of physical cells in blocks 1, 2, 4, and 5 along y-axis; block 3 has twice that number of cells along y;
- `xxl(iprt,1,1)` is the x-coordinate of the left end of part `iprt` along x-axis;
- `xxl(1,2,1)` is y-coordinate of the lower-left corner of block 1;
- `xxl(2,2,1)` is y-coordinate of the upper-left corner of block 5 (!)

Note that `xxl(2,2,1)` − `xxl(1,2,1)` is the y-extension of the entire mesh, and not just of its first block.

9.17. `igeom=2003`: a 5-block skewed Kershaw mesh in a rectangle with attached 5 blocks of orthogonal mesh

This is an extension of the previous case of a 5-block Kershaw mesh with an attached 5-block rectangular region along x-axis.

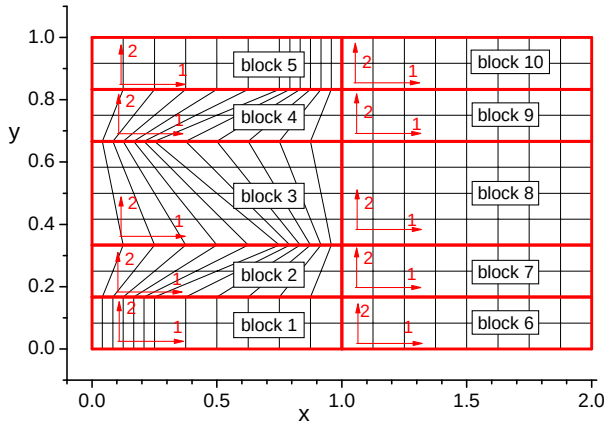


FIG. 9.17: A 5-block Kershaw mesh with an attached rectangle for `IGEOM` = 2003.

Identical boundary conditions are assigned along edges 3 of blocks 1–5, along edges 4 of blocks 6–10, along edges 1 of blocks 1 and 6, along edges 2 of blocks 5 and 10.

Mesh parameters for `igeom=2003`:

- **fixed:** `nblks` = 10
  `nprts(1,iblk)` = 2, `iblk` = 1–5,
  `nprts(1,iblk)` = 1, `iblk` = 6–10,
  `nprts(2,iblk)` = 1

- **user-must:** `ncell(1,1,1)`
  `ncell(1,2,1)`
  `xxl(1:3,1,1)`
  `xxl(1:2,2,1)`
  `ncell(1,1,6)`
  `xxl(2,1,6)`

- **user-can:**

`iradial`	def $= 0$
`fdx(1,1,6)`	def $= 1.0$
`ibc(1,1)`	def $= 2$
`ibc(3,1)`	def $= 2$
`ibc(4,6)`	def $= 2$
`ibc(2,5)`	def $= 2$
`ghwidth`	def $= 10^{-4}$

Explanations:

- `ncell(1,1,1)` is the number of physical cells in part 1 of the first five blocks along x-axis; part 2 contains equal number of cells;
- `ncell(1,2,1)` is the number of physical cells in blocks 1, 2, 4, 5, 6, 7, 9, 10 along y-axis; blocks 3 and 8 have twice that number of cells along y;
- `ncell(1,1,6)` is the number of physical cells along x-axis in blocks 6–10;
- `xxl(iprt,1,1)` is the x-coordinate of the left end of part `iprt` along x-axis;
- `xxl(1,2,1)` is y-coordinate of the lower-left corner of block 1;
- `xxl(2,2,1)` is y-coordinate of the upper-left corner of block 5 (!)
- `xxl(2,1,6)` is the x-coordinate of the right corners in blocks 6–10.

Note that `xxl(2,2,1)` − `xxl(1,2,1)` is the y-extension of the entire mesh, and not just of its first block.

9.18. `igeom=2005`: a single-block skewed Saltzman mesh in a rectangle

The number of blocks `nblks` = 1 is fixed; fixed also is the number of parts `nprts(m,1)` = 1. Parameter `iradial` is free to choose. The values of `fdx(iprt,m,iblk)` are irrelevant.

Mesh parameters for `igeom=2005`:

- **fixed:** `nblks=1`
  `nprts(m,1)=1`

- **user-must:** `ncell(1,m,1)`
  `xxl(1,m,1)`
  `xxl(2,m,1)`

- **user-can:**

`ibc(ib,1)`	def $= 2$
`ghwidth`	def $= 10^{-4}$

Explanations:

- `nprts(m,iblk)` is the number of parts along mesh direction `m` in part `iprt` of block `iblk`;
- `ncell(iprt,m,iblk)` is the number of physical cells along mesh direction `m` in part `iprt` of block `iblk`;
- `xxl(iprt,1,iblk)` and `xxl(iprt+1,1,iblk)` are the x coordinates of the left and right bounds of part `iprt` in block `iblk`;
- `xxl(iprt,2,iblk)` and `xxl(iprt+1,2,iblk)` are the y coordinates of the lower and upper bounds of part `iprt` in block `iblk`;
- `ghwidth` is the relative width of the ghost-cell bands;

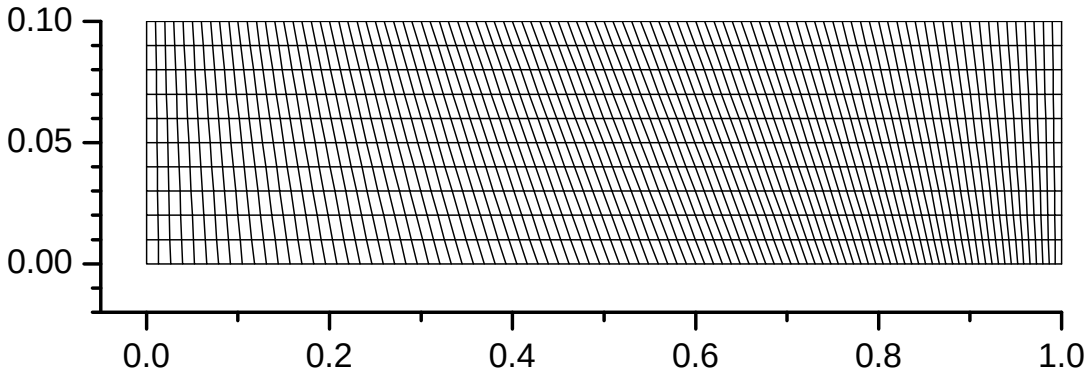


FIG. 9.18: A skewed Saltzman mesh in a rectangle.

9.19. `igeom=2201`: a 5-block "criss-cross" mesh in an arbitrary quadrangle

This is a special case of a 5-block skewed mesh in an arbitrary quadrangle; see Fig. 9.19.

Mesh parameters for `igeom=2201`:

- **fixed:** `nblks` $(= 5)$
- **user-must:** `ncell(iprt,m,1)`
  `ncell(iprt,m,2)`
  `x0aq(m,ic,1)` for `ic` $= 1, 2, 3, 4$
  `x0aq(m,ic,2)` for `ic` $= 2, 3, 4$
  `x0aq(m,ic,3)` for `ic` $= 2, 3$
  `x0aq(m,3,4)`

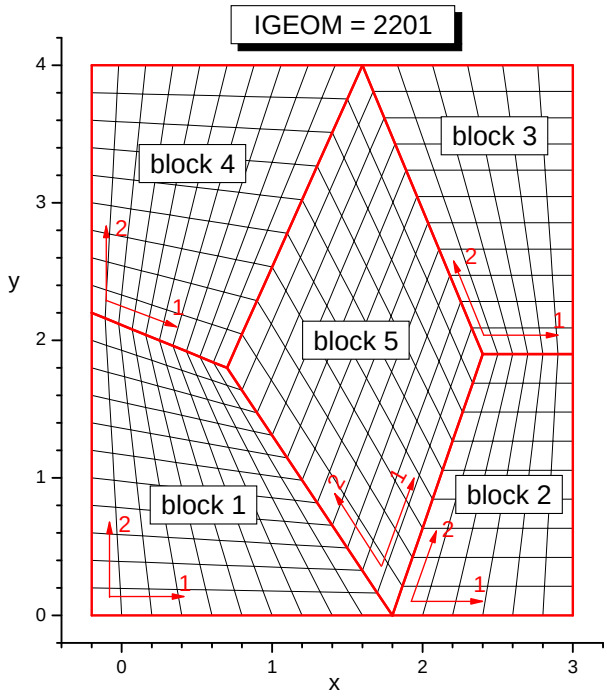


FIG. 9.19: A 5-block "criss-cross" mesh in an arbitrary quadrangle for `IGEOM` = 2201.

- **user-can:**

`iradial`	def $= 0$
`nprts(m,iblk)` for `iblk` $= 1, 2$	def $= 1$
`fprtaq(iprt,ib,iblk)` for `iblk` $= 1, 2$	def $= 1.0$
`fprtaq(iprt,ib,3)` for `ib` $= 2, 3, 4$	def $= 1.0$
`fprtaq(iprt,ib,4)` for `ib` $= 2, 3, 4$	def $= 1.0$
`fdxaq(iprt,ib,iblk)` for `iblk` $= 1, 2$	def $= 1.0$
`fdxaq(iprt,ib,3)` for `ib` $= 2, 3, 4$	def $= 1.0$
`fdxaq(iprt,ib,4)` for `ib` $= 2, 3, 4$	def $= 1.0$
`ibc(ib,1)` for `ib` $= 1, 3$	def $= 1.0$
`ibc(ib,2)` for `ib` $= 1, 4$	def $= 2$
`ibc(ib,3)` for `ib` $= 2, 4$	def $= 2$
`ibc(ib,4)` for `ib` $= 2, 3$	def $= 2$
`ghwidth`	def $= 10^{-4}$

The meaning of all the above mesh parameters is the same as for `igeom=22`.

9.20. `igeom=3001`: a multi-block distorted polar r-θ mesh

This version of a skewed polar mesh has been proposed in [**?**]; see Fig. 9.20. Each block is a circular sector with x_2 running along the radius r in the positive direction, and x_1 running along θ in the negative direction. The mesh parameters `x0(1,iblk)`, `dx(iprt,1,iblk)`, and `xxl(iprt,1,iblk)` should be assigned in angular degrees.

The meaning of all mesh parameters is the same as for `IGEOM` = 3 and 4. The mesh can be either uniform or progressive. As for `IGEOM` = 4, there are no fixed parameters. The

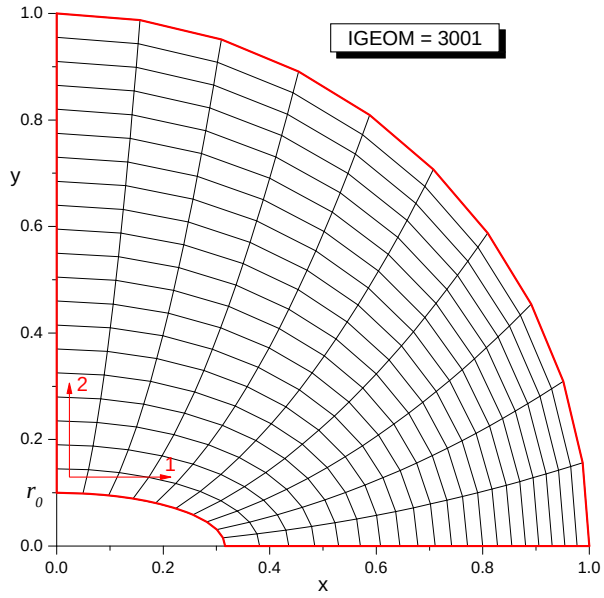


FIG. 9.20: A distorted polar mesh for `IGEOM` = 3001.

distortion is imposed by transformation

$$\begin{aligned} x_{ij} &= \sqrt{r_j}\,\cos\theta_i, \\ y_{ij} &= r_j\,\sin\theta_i, \end{aligned} \tag{9.70}$$

where

$$\begin{aligned} r_j &= r_0 + (j-1)\,\Delta r, \\ \theta_i &= \theta_0 + (i-1)\Delta\theta. \end{aligned} \tag{9.71}$$

9.21. `igeom=6001`: a quarter-circle cylindrical (or spherical) mesh with 3, 5, 7, ... blocks

This is a $(3+2k_t)$-block quarter-circle mesh for cylindrical and spherical configurations (depending on the value `iradial`), which has been been originally designed for tests. The central core inside the full circular domain is a square, represented by block 1; see Fig. 9.21. Above blocks 2 and 3 (each occupying 45°) there are k_t circular tiers, each comprised of 2 blocks. By setting reflective boundary conditions along the x- and y-axes, one can simulate half-circle or full-circle domains. This mesh is always progressive. The innermost corner (corner 1 in block 1) is always placed at `x0(1,1) =x0(2,1)` = 0.0.

Mesh parameters for ***`igeom=6001`****, progressive mesh:*

- **fixed:** `nprts(1,iblk)` = 1
  `nprts(2,iblk)` = 1, `iblk` = 1–3
  `ncell(1,m,iblk)` for `iblk` ≠ 2
  `ibc(ib,iblk)` (except for `ibc(1,1)`, `ibc(3,1)`, and `ibc(2,nblks-1)`)
  `nbc(ib,iblk)`

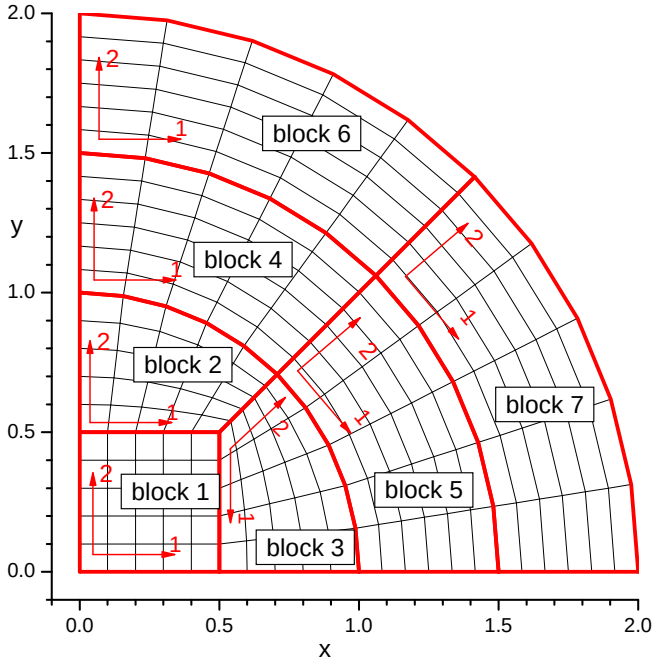


FIG. 9.21: A 7-block quarter-circle mesh.

- **user-must:**

```
ncell(1,1,1)
ncell(1,2,2)
ncell(1,2,2)
ncell(1:nprts(2,4),2,4)
ncell(1:nprts(2,6),2,6)
...
xxl(1,2,2)
xxl(2,2,2)
xxl(2:nprts(2,4)+1,2,4)
xxl(2:nprts(2,6)+1,2,6)
...
```

- **user-can:**

`iradial`	def $= 0$
`nblks`	def $= 3$
`nprts(2,4)`	def $= 1$
`nprts(2,6)`	def $= 1$
...	
`fdx(1,1,2)`	def $= 1.0$
`fdx(1,2,2)`	def $= 1.0$
`fdx(1:nprts(2,4),2,4)`	def $= 1.0$
`fdx(1:nprts(2,6),2,6)`	def $= 1.0$
...	
`ibc(1,1)`	def $= 2$
`ibc(3,1)`	def $= 2$
`ibc(2,nblks-1)`	def $= 2$
`ghwidth`	def $= 10^{-4}$

Explanations:

- `nprts(2,4)` is the number of parts along radial direction in blocks 4 and 5;
- `nprts(2,6)` is the number of parts along radial direction in blocks 6 and 7;
- ...
- `ncell(1,1,1)` is the number of physical cells along each direction in the central block 1;
- `ncell(1,2,2)` is the number of physical cells along radial direction in blocks 2 and 3;
- `ncell(kp,2,4)` is the number of physical cells in part `kp` along radial direction in blocks 4 and 5;
- ...
- `xxl(1,2,2)` is the edge length of the central square (block 1);
- `xxl(2,2,2)` is the outer radius of blocks 2 and 3;
- `xxl(kp+1,2,4)` is the outer radius of part `kp` in blocks 4 and 5;
- ...
- `fdx(1,1,2)` is the ratio of successive cell sizes along each mesh direction in the central square (block 1);
- `fdx(1,2,2)` is the ratio of successive cell sizes along the radial mesh direction in the outer blocks 2 and 3;
- `fdx(kp,2,4)` is the ratio of successive cell sizes along the radial mesh direction in part `kp` of blocks 4 and 5;
- ...
- `ibc(1,1)` is the type of boundary condition along the x-axis in blocks 1 and 3; typically `ibc(1,1)` $= 1$ or 2.
- `ibc(3,1)` is the type of boundary condition along the y-axis in blocks 1 and 2; typically `ibc(3,1)` $= 1$ or 2.
- `ibc(2,nblks-1)` is the type of boundary condition along the outer circular boundary; default value `ibc(2,nblks-1)=2`.

10. MESH REZONING AND REMAPPING OF PRINCIPAL VARIABLES

10.1. General scheme

By the end of the Lagrangian phase of the $(n+1)$-th hydrocycle known are the new values $m_i^{L,n+1} = m_i^n$ (cell masses), $\vec{u}_{c,i}^{\,L,n+1}$ (cell-centered fluid velocities), $e_i^{L,n+1}$ (mass-specific total energies) of the three principal hydro variables, which must be ascribed to mesh cells comoving with the fluid. Once the fluid velocities $\vec{u}_i$ at the mesh nodes are calculated in subroutine `LAGVEL`, the mesh is advanced from its old ALE configuration $\vec{x}_i^{\,n}$ to the new Lagrangian configuration as

$$\vec{x}_i^{\,L,n+1} = \vec{x}_i^{\,n} + \vec{u}_i \cdot \Delta t, \tag{10.1}$$

where Δt is the hydrodynamic time step of the $(n+1)$-th hydrocycle. For the pure Lagrangian mode ($\alpha_{ALE} = 1.0$) no additional rezoning/remapping is needed.

In the ALE and pure Eulerian modes ($0.0 \le \alpha_{ALE} < 1.0$) the new Lagrangian mesh $\vec{x}_i^{\,L,n+1}$ is reshaped (rezoned) to a new ALE configuration $\vec{x}_i^{\,n+1}$ in subroutine `REZONE` by applying an iterative algorithm based on the Winslow method, as described in the original CAVEAT report [1]. The resulting mesh displacement due to rezoning is expressed in terms of the velocity $\vec{u}_{r,i}$ of mesh motion, defined as

$$\vec{u}_{r,i} = (\vec{x}_i^{\,n+1} - \vec{x}_i^{\,L,n+1})/\Delta t. \tag{10.2}$$

Thus defined mesh-displacement velocities are used to perform the advection step, where the principal variables $m_i^{L,n+1}$, $\vec{u}_{c,i}^{\,L,n+1}$, $e_i^{L,n+1}$ are remapped (advected) from the Lagrangian mesh $\vec{x}_i^{\,L,n+1}$ to the new ALE mesh $\vec{x}_i^{\,n+1}$. The remapping is performed in subroutine `ADVECT` by using the fluxing volumes, calculated in subroutine `ADVFLUX`.

In the RALEF code the remapping can be done in one pass (i.e. with one step Δt in time), in two (i.e. with two fractional steps $0.5\Delta t$ in time) and in four (i.e. with four fractional steps $0.25\Delta t$ in time) passes, depending on whether the corresponding criterion is fulfilled; in the CAVEAT code no more than two passes were used. In each pass the method of directional splitting is applied, i.e. advection is first performed along mesh direction 1, and then along mesh direction 2 (or, alternatively, in the reverse order). The latter means that the advection step is actually split into 2, 4, or 8 substeps.

10.2. General features of the rezoning algorithm

The CAVEAT rezoning algorithm proved to be quite powerful and efficient in generating smooth and well adapted meshes — provided that the boundaries and internal material (i.e. Lagrangian) interfaces remain sufficiently smooth. In practice, however, often zigzag-like features develop along certain Lagrangian interfaces, which cause inversion of mesh directions and its eventual tangling. To cope with such situations, an additional smoothing algorithm, described in the next subsection, has been implemented in the RALEF code.

The general scheme of the driver routine `REZONE` for mesh rezoning is as follows:

- starting from the new Lagrangian mesh $\vec{x}_i^{\,L,n+1}$, an initial state $\vec{x}_i^{\,0,n+1}$ for rezoning is prepared in subroutine `REZINIT` by subtracting the fraction $(1 - \alpha_{ALE})$ of the coordinate increment on the right-hand side of Eq. (10.1); along the Lagrangian interfaces, this subtraction is performed tangentially; nodes on the inflow/outflow boundaries

are returned to these boundaries on the assumption that the latter are given by surfaces that are fixed in space; the cell-centered weight coefficients w_i are computed in subroutines `WEIGHT` and `WTGLOBL`;

- block edges marked by the values `ibcrezn(ib,iblk)` (mod 10) > 0 are subject to a special smoothing procedure (as described below) via calls to subroutines `REZSMOO` and/or `REZSTRE`; subroutine `REZSTRE` is a slight modification of the original `REZTANG` routine from CAVEAT; as a result, the initial state $\vec{x}_i^{\,0,n+1}$ is modified to $\vec{x}_i^{\,1,n+1}$;

- rezoning coefficients `arzn(i,1:3)`, `brzn(i,1:2)` are calculated in subroutine `REZCOEF` for all physical vertices by using the initial mesh configuration $\vec{x}_i^{\,1,n+1}$ and the weights w_i;

- an intermediate rezoned mesh $\vec{x}_i^{\,w,n+1}$ is calculated by performing `itrezn` iterations on the initial state $\vec{x}_i^{\,1,n+1}$ in subroutine `REZITER`; at each iteration the belts of ghost cells with a 100% relative width must be reconstructed; also, a special action is needed at 3-block meeting points to equalize the coordinates of the corresponding three block corners, which is not ensured by the adopted iteration algorithm and is accomplished in subroutine `BCRNRMRG`;

- the final rezoned mesh configuration

$$\vec{x}_i^{\,n+1} = \vec{x}_i^{\,L,n+1} + \min\left\{1; \frac{a_{mdx} d_i}{\left|\vec{x}_i^{\,w,n+1} - \vec{x}_i^{\,L,n+1}\right|}\right\}\left(\vec{x}_i^{\,w,n+1} - \vec{x}_i^{\,L,n+1}\right) \tag{10.3}$$

is obtained by constraining the values $\vec{x}_i^{\,w,n+1}$ in subroutine `REZLIM` in such a way as to ensure that the mesh displacement $\left|\vec{x}_i^{n+1} - \vec{x}_i^{\,L,n+1}\right|$ is no larger than a_{amdx} times the distance d_i to the nearest of the four neighbor vertices. Parameter a_{mdx} is assigned a fixed value of $a_{mdx} = 0.7$ in subroutine `REZLIM`. In order to eliminate possible deviations from spatially fixed boundaries, corresponding boundary conditions on reflecting, center-of-convergence, and inflow/outflow boundaries are imposed on the values of $\vec{u}_{r,i}$ in subroutine `BNDUMRZ`.

10.3. Algorithm for smoothing corrugated interfaces

In cases, where tangential rezoning tends to produce unwanted numerical corrugation of Lagrangian interfaces, it is proposed to use the following special smoothing procedure. Let $\vec{x}_i$ be the original coordinates of vertices along an interface to be smoothed. Then, the new coordinates of the smoothed interface are calculated as

$$\vec{x}_{sm,i} = \vec{x}_i + c_{sm}\left(\vec{x}_{s,i} - \vec{x}_i\right), \tag{10.4}$$

where $\vec{x}_{s,i}$ are the "ideal" (desired) coordinates of vertex i, and c_{sm} is a free scaling parameter on the order of unity. The ideal location $\vec{x}_{s,i}$ of vertex i is determined by a symmetric local algorithm, based on a 5-point stencil $\{\vec{x}_{i-2}, \vec{x}_{i-1}, \vec{x}_i, \vec{x}_{i+1}, \vec{x}_{i+2}\}$.

As a first step, we calculate the turn angle θ_i at each vertex i (see Fig. 10.1) as

$$\theta_i = \mathrm{ATAN2}(\Delta y_i, \Delta x_i), \tag{10.5}$$

where

$$\Delta x_i \;=\; (\vec{x}_{i+1} - \vec{x}_i) \cdot (\vec{x}_i - \vec{x}_{i-1}), \tag{10.6}$$
$$\Delta y_i \;=\; (\vec{x}_{i+1} - \vec{x}_i) \times (\vec{x}_i - \vec{x}_{i-1}). \tag{10.7}$$

Here θ_i is the rotation angle of segment $\vec{x}_{i+1} - \vec{x}_i$ with respect to the preceding segment $\vec{x}_i - \vec{x}_{i-1}$ (in the counterclockwise direction) in radians; the FORTRAN intrinsic function ATAN2(y, x) yields the principal value of the argument of a complex number $x + iy$.

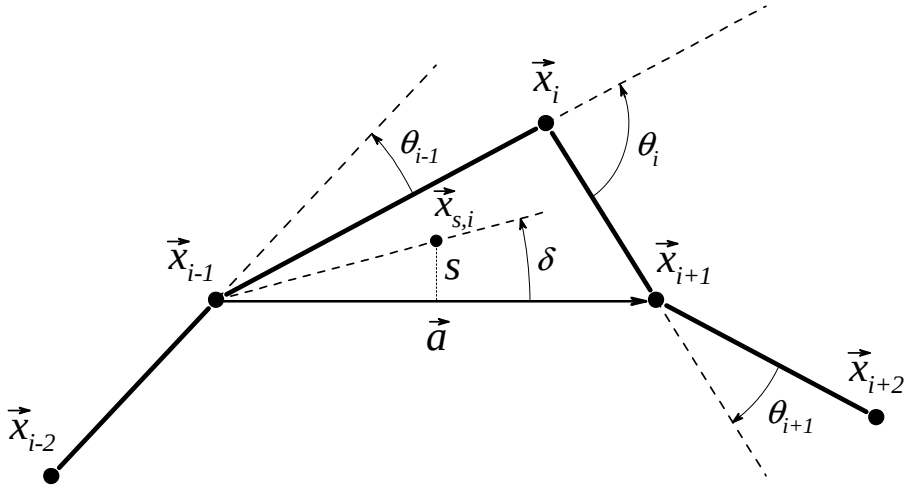


FIG. 10.1: Rezoning scheme for smoothing a corrugated interface.

Next, we assume that the ideal location $\vec{x}_{s,i}$ of vertex i would correspond to a uniform distribution of the five stencil vertices $\{\vec{x}_{i-2}, \vec{x}_{i-1}, \vec{x}_i, \vec{x}_{i+1}, \vec{x}_{i+2}\}$ along a common circumscribing circle. In such an ideal case vertex i would lie on a normal $\vec{a}_*$ to the segment $\vec{a} = \vec{x}_{i+1} - \vec{x}_{i-1}$, passing through its midpoint $\frac{1}{2}(\vec{x}_{i+1} + \vec{x}_{i-1})$, at a distance

$$s = \frac{1}{2} \, |\vec{a}| \cdot \tan \delta \tag{10.8}$$

from this midpoint, i.e. we would have

$$\vec{x}_{s,i} = \frac{1}{2} \, (\vec{x}_{i-1} + \vec{x}_{i+1}) + \frac{s}{|\vec{a}|} \, \vec{a}_*, \tag{10.9}$$

where the vectors $\vec{a}$ and $\vec{a}_*$ are defined as

$$\vec{a} \equiv \{a_x, a_y\} = \vec{x}_{i+1} - \vec{x}_{i-1}, \quad \vec{a}_* = \{-a_y, a_x\}, \tag{10.10}$$

and the angle δ would be given by

$$\delta = \frac{1}{2}\theta_i = \frac{1}{2}\theta_{i-1} = \frac{1}{2}\theta_{i+1}. \tag{10.11}$$

In a real non-ideal case of an arbitrary polygonal line, where the turn angles $\theta_{i-1} \neq \theta_i \neq \theta_{i+1}$, we can use the average

$$\delta = \frac{1}{6} \, (\theta_{i-1} + \theta_i + \theta_{i+1}) \tag{10.12}$$

as the best approximation to the desired value of angle δ. As a result, we arrive at the following smoothing correction to the coordinates of vertex i

$$\vec{\tilde{x}}_i = \vec{x}_i + c_{sm}\left[\frac{1}{2}\left(\vec{x}_{i-1} + \vec{x}_{i+1}\right) - \vec{x}_i + \frac{1}{2}\,\vec{a}_*\,\tan\delta\right], \tag{10.13}$$

where δ is given by Eq. (10.12).

To obtain an a priori estimate for the coefficient c_{sm}, we may consider a simple particular case of all vertices $\vec{x}_i$ lying on a straight line but being non-uniformly spaced. In this case $\delta = 0$ because all $\theta_i = 0$, and our smoothing procedure will push the vertices toward a more uniform spacing along the interface. If, say, we consider a distribution that is extremely non-uniform around one point $x = 0$ with vertices $\ldots, x_{-2} = -2$, $x_{-1} = -1$, $x_0 = x_1 = 0$, $x_2 = 1$, $x_3 = 2, \ldots$, then after a single smoothing iteration we would like to have the values close to $x_0 = -1/3$, $x_1 = +1/3$. One immediately verifies that the latter is obtained from Eq. (10.13) with $c_{sm} = 2/3$. For practical applications, to avoid overshooting, somewhat smaller values of $c_{sm} = 0.4$–0.5 can be recommended. Figure 10.2 shows an example of application of the above smoothing algorithm to a strongly distorted boundary, where further simulation without such smoothing would have been impossible.

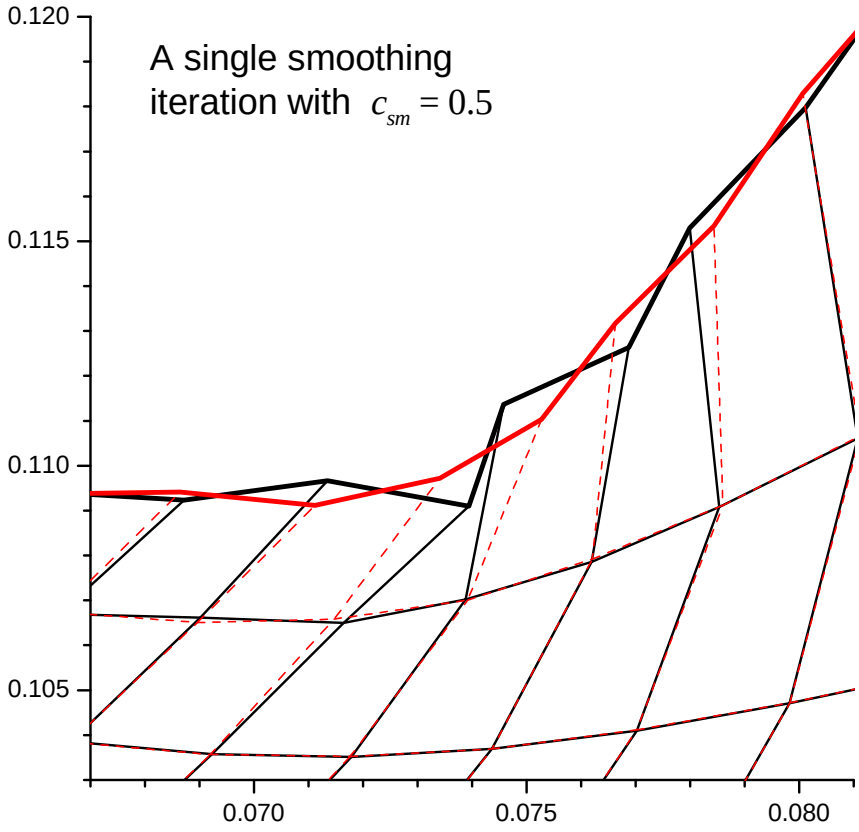


FIG. 10.2: Example of application of a single smoothing iteration with $c_{sm} = 0.5$ to a corrugated external boundary.

It should be noted that, strictly speaking, the above described rezoning algorithm does not conserve the area of the region, bounded by the rezoned interface. However, due to high

order of approximation, the inflicted changes in this area are quite small and usually have no practical significance — which is clearly seen in Fig. 10.2.

Practical tests show that unconditional application of the smoothing algorithm to selected interfaces at every time step may sometimes significantly slow down simulation not because of an extra time required by the smoothing algorithm itself but because too fine and/or skewed mesh is generated at certain locations. To circumvent this undesirable property, additional conditions are introduced into the algorithm that are checked before applying the smoothing procedure to a selected interface. These conditions are formulated as two criteria controlled by the value of parameter p_{sm}: for $p_{sm} > 1$ criterion 1 is applied, whereas for $0 < p_{sm} < 1$ criterion 2 is applied; for $p_{sm} = 1$ the smoothing is performed unconditionally at every time step.

According to **criterion 1**, the entire interface is subject to a smoothing iteration if for at least one of its vertices either

$$|\theta_i| > 0.8, \tag{10.14}$$

or

$$3\left(\theta_{i-1}^2 + \theta_i^2 + \theta_{i+1}^2 - 3\theta_0^2\right) > p_{sm}\left(\theta_{i-1} + \theta_i + \theta_{i+1}\right)^2, \tag{10.15}$$

or

$$\max\left(|\vec{x}_i - \vec{x}_{i-1}|, |\vec{x}_{i+1} - \vec{x}_i|\right) > (2p_{sm} - 1)\min\left(|\vec{x}_i - \vec{x}_{i-1}|, |\vec{x}_{i+1} - \vec{x}_i|\right). \tag{10.16}$$

According to **criterion 2**, the entire interface is subject to a smoothing iteration if for at least one of its vertices either

$$|\theta_i| > 0.8, \tag{10.17}$$

or

$$\theta_{i-1} \cdot \theta_i < 0 \ \text{ and } \ \theta_{i-1} \cdot \theta_{i+1} > 0 \ \text{ and } \ \min\left(|\theta_{i-1}|, |\theta_i|, |\theta_{i+1}|\right) > p_{sm}. \tag{10.18}$$

Criterion 1 contains an additional free parameter: a sensitivity threshold θ_0 for the turn angles. The larger the values of p_{sm} and θ_0, the weaker the criterion, and the less frequently is the smoothing procedure applied. High quality smooth interfaces without adverse side effects are obtained for $p_{sm} = 1.5$–2 and $\theta_0 = 0.02$–0.04. Criterion 2 is generally less restrictive than criterion 1; good results are usually obtained for $p_{sm} = 0.01$–0.1.

All block corners are excluded from the smoothing procedure. Also excluded (because of the 5-point stencil) are the next neighbors to the block corners unless the second meeting block edge at the corresponding corner is either a reflective or an interblock boundary — for which ghost cells have "real" sizes.

Application of the described smoothing algorithm is realized by calling the subroutine `REZSMOO` and is controlled by the value of flag `ibcrezn(ib,iblk)`: the criteria for application of the algorithm to an edge `ib` of block `iblk` are tested only when `ibcrezn(ib,iblk)` $(\text{mod } 10) = 1, 2, 4, 5, \ldots, 9$.

10.4. Algorithm for pocket filling where an interface tends to fold up

Because the above described smoothing algorithm is local (based on a 5-point stencil along one dimension), it cannot prevent formation of deep bottle-shaped pockets, which encompass significantly more than 5 cell faces, and where the interface eventually folds up and mesh discretization of the computational domain becomes multivalued; see Fig. 10.3. To straighten up such a pocket forming at a Lagrangian interface before it closes up in the neck area, the following "pocket-fill" algorithm has been implemented.

First of all, formation of a potentially dangerous pocket is probed by summing up the turn angles θ_{i-j} and θ_{i+j}, $j = 0, 1, 2, \ldots, n_{bay} + k_{jt} - 1$, for $n_{bay} + k_{jt} - 1$ vertices "down-stream" and "up-stream" from a considered vertex i, and finding the two maximum values of the

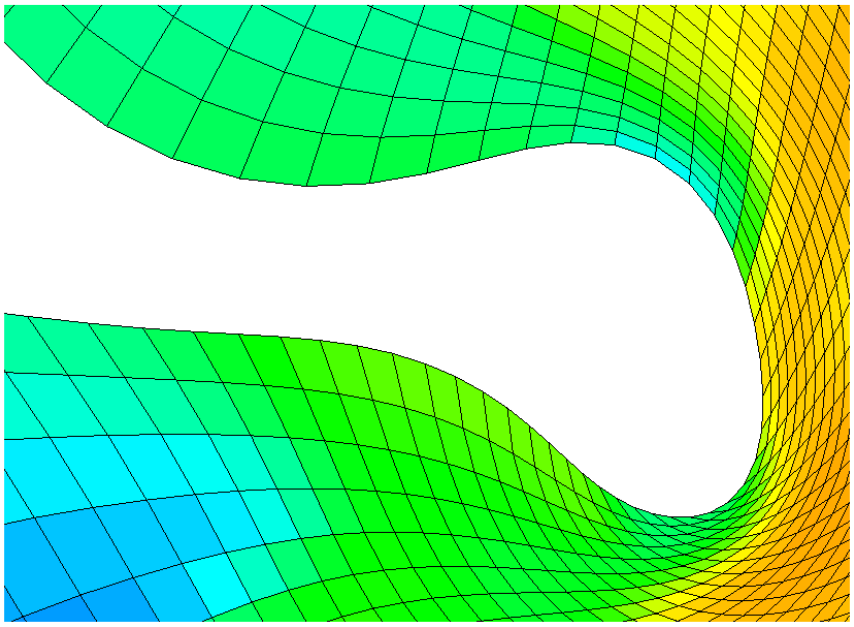


FIG. 10.3: Example of a bottle-shaped vacuum pocket along a corrugated external boundary.

accumulated turn angle. In other words, for each vertex i we calculate the two maximum accumulated turn angles as

$$\Delta\theta^{-}_{m,i} = \max_{k}\left\{-\sum_{j=0}^{k}\delta_{ed}\theta_{i-j}\right\}, \qquad \Delta\theta^{+}_{m,i} = \max_{k}\left\{-\sum_{j=0}^{k}\delta_{ed}\theta_{i+j}\right\}, \tag{10.19}$$

where $k = 0, 1, 2, \ldots, n_{bay} + k_{jt} - 1$, and the factor

$$\delta_{ed} = \texttt{iend(ib)} = \begin{cases} +1, & \text{for the left and top block edges } (\texttt{ib} = 2, 3), \\ -1, & \text{for the bottom and right block edges } (\texttt{ib} = 1, 4) \end{cases} \tag{10.20}$$

indicates the mesh direction towards the primary corner of block edge `ib`. Parameter n_{bay} is user defined (typical values are $n_{bay} = 6$–12), whereas the "jitter" shift k_{jt} — which takes on the values $k_{jt} = \texttt{ncyc} \pmod 3 = 0$, 1, or 2 — is introduced with the purpose to avoid sharp corners at a distance $\pm n_{bay}$ from the bottom of the filled pocket. If both $\Delta\theta^{-}_{m,i}$ and $\Delta\theta^{+}_{m,i}$ are positive and large, we have a vacuum pocket with a bottom near vertex i (if the block edge `ib` does border vacuum) — as it is shown in Fig. 10.3.

Once the quantities $\Delta\theta^{\pm}_{m,i}$ have been determined, they are compared with the threshold value θ_* (a user-defined parameter), and, if the condition

$$\theta_{bay} \equiv \min\{\Delta\theta^{-}_{m,i},\, \Delta\theta^{+}_{m,i}\} > \theta_* \tag{10.21}$$

is fulfilled, then a "pocket-fill" shift is applied to vertex i. In practice, good results are usually obtained with $\theta_* = 0.6$–1.2.

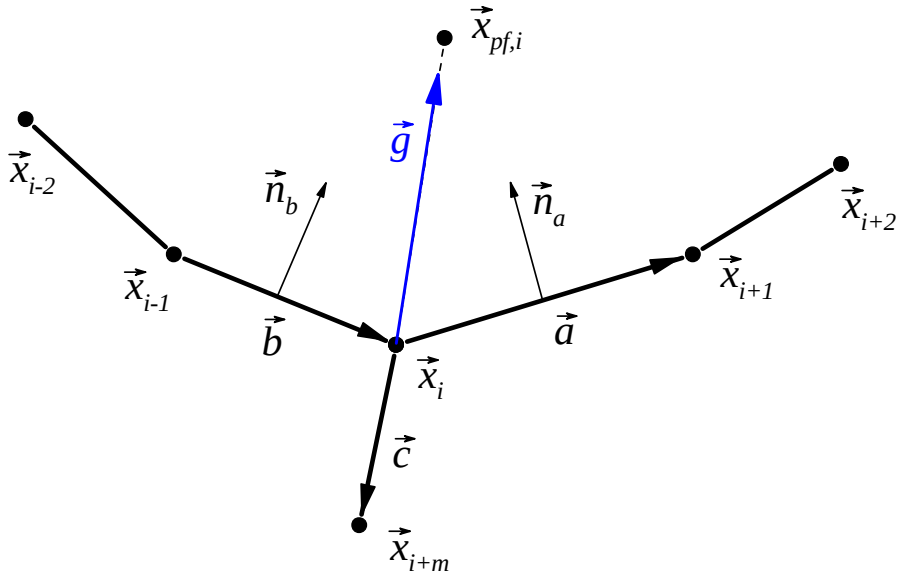


FIG. 10.4: Notation scheme for the "pocket-fill" shift of vertex i.

In the process of the "pocket-fill" shift the old position $\vec{x}_i$ of vertex i is shifted to a new one $\vec{x}_{pf,i}$ along a vector

$$\vec{g} = \delta_{ed}\,(\vec{n}_a + \vec{n}_b)\,, \tag{10.22}$$

where

$$\vec{a} \equiv \{a_x, a_y\} = \vec{x}_{i+1} - \vec{x}_i, \quad \vec{n}_a = \left\{ \frac{-a_y}{\sqrt{a_x^2 + a_y^2}}, \frac{a_x}{\sqrt{a_x^2 + a_y^2}} \right\}, \tag{10.23}$$

$$\vec{b} \equiv \{b_x, b_y\} = \vec{x}_i - \vec{x}_{i-1}, \quad \vec{n}_b = \left\{ \frac{-b_y}{\sqrt{b_x^2 + b_y^2}}, \frac{b_x}{\sqrt{b_x^2 + b_y^2}} \right\}; \tag{10.24}$$

see Fig. 10.4. The magnitude of this shift is calculated as

$$\vec{x}_{pf,i} = \vec{x}_i + \alpha_{bay}\, \min\{|\vec{a}|,\, |\vec{b}|,\, |\vec{c}|\}\, \frac{\vec{g}}{|\vec{g}|}, \tag{10.25}$$

where

$$\alpha_{bay} = c_{bay}\, h_{bay}\, h_\theta, \tag{10.26}$$

$$h_{bay} = \min \left\{ 1,\, \max \left[0,\, \delta_{ed} \frac{\vec{a}_{bay} \times \vec{b}_{bay}}{\left|\vec{b}_{bay}\right|^2} \right] \right\}, \tag{10.27}$$

$$h_\theta = \min \left\{ 1,\, \max \left[0,\, \frac{\theta_{bay} - \theta_*}{\pi/2 - \min(1.4,\, \theta_*)} \right] \right\}. \tag{10.28}$$

Here

$$\vec{c} = \vec{x}_{i+m} - \vec{x}_i \tag{10.29}$$

is a vector connecting vertex i to its neighbor $i + m$ inside the considered block along the mesh direction, other than the mesh direction from $i-1$ to i to $i+1$ (see Fig. 10.4); $c_{bay} \leq 0.8$

is the user defined amplitude of the "pocket-fill" coordinate shift; h_{bay} is the dimensionless depth of the forming pocket; h_θ is a relative measure of how concave the forming pocket is; vector

$$\vec{a}_{bay} = \vec{x}_i - \vec{x}_{im-} \tag{10.30}$$

connects vertex $im- = i - k_{m-}$, where the maximum $\Delta\theta^-_{m,i}$ [see Eq. (10.19)] is reached, with vertex i; vector

$$\vec{b}_{bay} = \vec{x}_{im+} - \vec{x}_{im-} \tag{10.31}$$

connects vertex $im-$ with the vertex $im+ = i + k_{m+}$, where the maximum $\Delta\theta^+_{m,i}$ is reached; see Fig. 10.5.

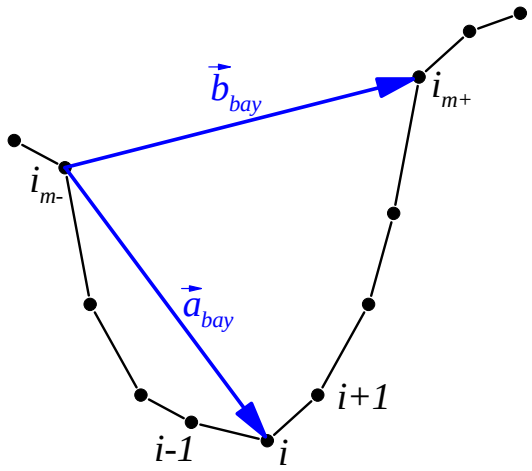


FIG. 10.5: Vectors $\vec{a}_{bay}$ and $\vec{b}_{bay}$ used to evaluate the relative depth of a vacuum pocket around vertex i.

TABLE III: The values of the flag array `ibcrezn(ib,iblk)` used to control application of the smoothing, "pocket-fill", and stretching algorithms to an edge `ib` of block `ib`.

`ibcrezn(ib,iblk)` (mod 10)	smoothing	pocket-fill	stretching
0	no	no	no
1	yes	no	no
2	yes	yes	no
3	no	no	yes
4	yes	no	yes
5	yes	yes	yes
≥ 6	yes	no	no

The "pocket-fill" algorithm is applied always in combination with the smoothing algorithm of subsection 10 10.3 by calling the subroutine `REZSMOO`, and it always precedes the smoothing algorithm. The latter means that, if the vertex coordinates $\vec{x}_i$ along a certain interface were modified to $\vec{x}_{pf,i}$ by the "pocket-fill" algorithm, it is the modified values $\vec{x}_{pf,i}$ rather than $\vec{x}_i$ that serve as an input to Eq. (10.4) in the smoothing algorithm; for this, the values of the turn angles θ_i in Eq. (10.5) used in the smoothing algorithm are recalculated

for the modified interface configuration $\vec{x}_{pf,i}$. To avoid too strong mesh deformations, the "pocket-fill" algorithm is applied no more frequently than every second hydro cycle. Application of the "pocket-fill" and the smoothing algorithms to an edge `ib` of block `iblk` is controlled via user defined values to the flag `ibcrezn(ib,iblk)`, as it is explained in Table III.

10.5. Second-order remapping

In the RALEF code the spatial order of the remapping scheme is controlled by a separate user-defined parameter `iordadv`. If the user specifies `iordadv` = 1 (or 2), the first-order (or second-order) advection scheme is used independent of the value of the control parameter `iorder` for the Riemann solver. If `iordadv` remains unspecified by the user, its value is set equal to that of `iorder` (the original CAVEAT option).

1. Gradient limiting

The second-order remapping scheme uses the cell-centered gradients of three volume-specific quantities — namely, of the density ρ, the volume-specific momentum $\rho\vec{u}$, and the volume specific total energy $\rho e = \rho\epsilon + \frac{1}{2}\rho u^2$ — that are calculated in the subroutine `GRADIENT`. An important role belongs to the method of limiting the calculated gradients, which helps to avoid unphysical values of remapped quantities near sharp fronts.

The method of gradient limiting is controlled by the user-defined parameter `limgrad`. In the RALEF code the following options are available:

`limgrad` = −1 :	the unlimited vertex-centered gradients are calculated for all quantities in subroutine `GRADIENT`;
`limgrad` = 0 :	the method of monotonic limiting is applied to all fields: the limited cell-centered gradient is the minimum of the 4 gradients at 4 vertices if they all are positive, the maximum of the 4 if they all are negative, and zero otherwise; all the gradients calculated in subroutine `GRADIENT` are cell-centered;
`limgrad` = +1 :	the *regular van Leer* limiting method is applied to the scalar fields (density, pressure, ...), while the gradients of the vector fields (velocity, momentum, ...) are left unlimited; all the gradients calculated in subroutine `GRADIENT` are cell-centered;
`limgrad` = +2 :	the *regular van Leer* limiting method is applied to all fields; all the calculated gradients are cell-centered;
`limgrad` = +3 :	the *extended van Leer* limiting method is applied to all fields; all the calculated gradients are cell-centered.

The original CAVEAT code had only the monotonic and the regular van Leer gradient limiting, employed with the options `limgrad` = 1 and 2. The extended van Leer limiting with the option `limgrad` = 3 is a new development in the RALEF code implemented in November 2011.

When a cell-centered gradient $\vec{g}_i$ of a quantity q is calculated (see [1, §3.1.2]), the latter means that the spatial variation of q across cell i is approximated by a linear function

$$\tilde{q}(\vec{x}) = q_i + \vec{g}_i \cdot (\vec{x} - \vec{x}_{c,i}), \tag{10.32}$$

where q_i is the value of q at the center of cell i with coordinates $\vec{x}_{c,i}$. When the van Leer method of gradient limiting is applied, the gradient $\vec{g}_i$ in Eq. (10.32) is replaced by the

product $\alpha_i \vec{g}_i$, where the limiting factor $0 \le \alpha_i \le 1$ is given by

$$\alpha_i = \min\{1,\, \alpha_{max},\, \alpha_{min}\}, \tag{10.33}$$

with

$$\alpha_{max} = \max\left\{0,\, \frac{q_{8n,max} - q_i}{\tilde{q}_{max} - q_i}\right\}, \qquad \alpha_{min} = \max\left\{0,\, \frac{q_{8n,min} - q_i}{\tilde{q}_{min} - q_i}\right\}. \tag{10.34}$$

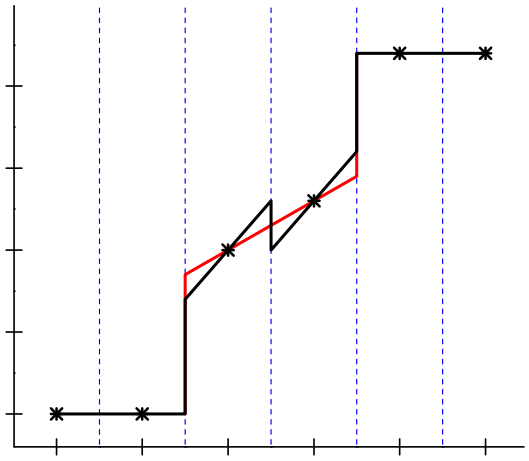


FIG. 10.6: The regular van Leer (black) and the extended van Leer (red) methods of gradient limiting.

In the above equations

$$q_{8n,max} = \max_{l=1,2,\ldots,8}\{q_{i+k_{8,l}}\}, \qquad q_{8n,min} = \min_{l=1,2,\ldots,8}\{q_{i+k_{8,l}}\}, \tag{10.35}$$

are, respectively, the maximum and the minimum values of q_i from the 8 neighboring cells surrounding the given cell i, which participate in calculating the gradient $\vec{g}_i$; $k_{8,l}$ are the local index offsets with respect to i for these 8 neighbors. In the *regular van Leer* limiting algorithm (`limgrad=1,2`) the maximum, $\tilde{q}_{max}$, and the minimum, $\tilde{q}_{min}$, values of the linear function (10.32) (with unlimited gradients $\vec{g}_i$) are calculated by only considering its variation *inside* the cell i, i.e. among the four corner values

$$\tilde{q}_{max} = \max_{l=1,2,3,4}\{\tilde{q}(\vec{x}_{i+k_{4,l}})\}, \qquad \tilde{q}_{min} = \min_{l=1,2,3,4}\{\tilde{q}(\vec{x}_{i+k_{4,l}})\}, \tag{10.36}$$

where $k_{4,l}$ are the local index offsets with respect to i for the 4 corner vertices of cell i, and $\vec{x}_i$ are the coordinates of cell vertex i.

As mentioned in the original CAVEAT report [1], the regular van Leer limiting sometimes produces locally non-monotonic sawtooth-like variation of the approximated quantity $q(\vec{x})$ at cell interfaces, as is illustrated with the black polygonal line in Fig. 10.6. Quite often this violation of local monotonicity becomes prohibitive for using the second-order in the advection scheme. As a simple and effective remedy, the following modification of the regular van Leer limiting has been introduced in the RALEF code: when calculating the quantities

$\tilde{q}_{max}$ and $\tilde{q}_{min}$, the domain of variation of the linear interpolation (10.32) must be extended to include the centers of the 8 surrounding neighbor cells, i.e. to modify Eq. (10.36) to

$$\begin{aligned} \tilde{q}_{max} &= \max_{l=1,2,3,4;\, l'=1,2,\ldots,8} \{\tilde{q}(\vec{x}_{c,i+k_{4,l}}),\ \tilde{q}(\vec{x}_{i+k_{8,l'}})\}, \\ \tilde{q}_{min} &= \min_{l=1,2,3,4;\, l'=1,2,\ldots,8} \{\tilde{q}(\vec{x}_{i+k_{4,l}}),\ \tilde{q}(\vec{x}_{c,i+k_{8,l'}})\}. \end{aligned} \tag{10.37}$$

As a result, we obtain a more restrictive *extended van Leer* method of gradient limiting, illustrated in Fig. 10.6 with the red polygonal line.

It is easy to see that for 1D interpolation the extended van Leer method is in fact equivalent to the monotonic limiting (`limgrad=0`) already implemented in the original CAVEAT code — which, however, is not the case in two and three dimensions! Practical tests have demonstrated that in many cases, where both the regular van Leer and the monotonic gradient limiting algorithms fail (resulting, typically, in negative temperatures and/or unphysical reduction of the hydrodynamic time step by many orders of magnitude), the extended van Leer limiting facilitates a smooth simulation with the second-order advection scheme. Therefore, the default value of the `limgrad` parameter is set equal to `limgrad=3`.

2. Limits on the 2nd-order advection terms

The advection scheme inherited from the CAVEAT code is conservative with respect to the three principal advected variables: mass, momentum and total (kinetic + internal) energy. The internal energy is then calculated as the difference between the total and the kinetic energy components. Clearly, in certain cases this may lead to spurious unphysical drops in the value of the internal energy, accompanied by unphysical reduction of temperature (often below zero) and practical stoppage of the simulation. Because the total energy is conserved, we can call such behavior a *negative advective dissipation*, or consider it as a manifestation of a *negative numerical viscosity*.

The key quantity for computing the amount of advection in cell i for any of the principal variables at every advection substep is the *fluxing volume* V^a_{im}, associated with face im. The sign of V^a_{im} is defined such that the change in volume of cell i is given by

$$V_i = V_i^L + V^a_{im}. \tag{10.38}$$

Hence, if at a given advection substep face $i1$ ($i2$) is displaced as a whole in the positive mesh direction 2 (1) along the normal $\vec{n}_{f,i1}$ ($\vec{n}_{f,i2}$), we have $V^a_{im} < 0$ because such a displacement results in the reduction of the volume V_i of cell i; see Fig. 8.1.

First of all, consider the core part of all the advection terms, which is obtained in the 1-st order with the full donor treatment. The extent of the donor-cell treatment is controlled by the user defined parameter $0 \le f_d \le 1$: f_d is the weight coefficient for the donor-cell value of the advected quantity, $1 - f_d$ is the weight coefficient for the acceptor-cell value of the advected quantity. In the case of the full donor treatment we have $f_d = 1.0$, which means a 100% upstream advection.

Here we prove the following statement:

> *the artificial advective dissipation of the kinetic energy is always non-negative in the 1-st order with the full (100% upstream) donor treatment.*

Indeed, it is quite clear that for $V^a_{im} < 0$ we have zero advective dissipation in cell i because the donor cell is the cell i itself, and the reductions in mass, momentum, total, kinetic and internal energies of cell i are all strictly proportional to the fractional reduction of its volume (i.e. are all altered in one and the same proportion).

Now consider a less evident case of $V_{im}^a > 0$. Here the cell i receives a certain amount of mass

$$\Delta m = \rho_{i_d} V_{im}^a > 0 \tag{10.39}$$

from the neighbor (across face im) donor cell $i_d = i_o$ (an equal amount of mass is, of course, lost by the donor cell). With this mass the acceptor cell i receives also the amount of momentum

$$\Delta \vec{P} = \Delta m \, \vec{u}_{i_d}, \tag{10.40}$$

and the amount of total energy

$$\Delta E = \Delta m \, e_{i_d} = \Delta m \left(\epsilon_{i_d} + \frac{1}{2} u_{i_d}^2 \right), \tag{10.41}$$

where $\vec{u}_{i_d}$ is the cell-center fluid velocity in the donor cell i_d, e_{i_d} is its mass-specific total energy, and ϵ_{i_d} is its mass-specific internal energy.

Because the internal energy $\mathcal{E}_i$ of any cell i is calculated as

$$\mathcal{E}_i = E_i - \frac{P_i^2}{2m_i}, \tag{10.42}$$

where E_i is its total energy, and P_i is its total momentum, the advective increment of the internal energy in cell i is given by

$$\Delta \mathcal{E} = \Delta E + \frac{P_i^2}{2m_i} - \frac{\left(\vec{P}_i + \Delta \vec{P} \right)^2}{2(m_i + \Delta m)}. \tag{10.43}$$

To find the amount of artificial advective dissipation, we have to compare (10.43) with the direct advective increment of the internal energy

$$\Delta \mathcal{E}_0 = \Delta m \, \epsilon_{i_d}, \tag{10.44}$$

which cell i would have received with mass Δm in the absence of any dissipation. As a result, the amount of artificial kinetic energy dissipation due to the first-order upstream advection across face im is found to be

$$d\mathcal{E}_a \equiv \Delta \mathcal{E} - \Delta \mathcal{E}_0 = \frac{m_i \, \Delta m}{2(m_i + \Delta m)} \left(\vec{u}_{i_d} - \vec{u}_i \right)^2 \geq 0. \tag{10.45}$$

Thus, negative advective dissipation may arise only due to a partial donor treatment ($f_d < 1.0$) and/or second-order terms. In this general case the advection terms Δm, $\Delta \vec{P}$, and ΔE, associated with face im and added to cell i, are calculated as follows. Let i_o be the neighbor cell of cell i, which lies across the face im. Then the donor cell i_d and the acceptor cell i_a are determined as

$$i_d = \begin{cases} i, & V_{im}^a < 0, \\ i_o, & V_{im}^a \geq 0, \end{cases} \qquad i_a = \begin{cases} i, & V_{im}^a \geq 0, \\ i_o, & V_{im}^a < 0. \end{cases} \tag{10.46}$$

The principal advection terms are given by the sum of two components

$$\Delta m = \Delta m^{(1)} + \lambda \, \Delta m^{(2)}, \tag{10.47a}$$

$$\Delta \vec{P} = \Delta \vec{P}^{(1)} + \lambda \, \Delta \vec{P}^{(2)}, \tag{10.47b}$$

$$\Delta E = \Delta E^{(1)} + \lambda \, \Delta E^{(2)}, \tag{10.47c}$$

where

$$\Delta m^{(1)} = \rho_{i_d} V_{im}^a, \tag{10.48a}$$

$$\Delta \vec{P}^{(1)} = \rho_{i_d} \vec{u}_{i_d} V_{im}^a = \vec{u}_{i_d} \Delta m^{(1)}, \tag{10.48b}$$

$$\Delta E^{(1)} = \rho_{i_d} e_{i_d} V_{im}^a = e_{i_d} \Delta m^{(1)}, \tag{10.48c}$$

are the first-order full-donor contributions, whose numerical dissipation is always positive; also, for a non-inverted mesh, $\Delta m^{(1)}$ cannot lead to a negative mass of either donor or acceptor cell.

The second components in Eq. (10.47),

$$\Delta m^{(2)} = \left[\Delta\rho^{(12)} + \Delta\rho^{(2)}\right] V_{im}^a, \tag{10.49a}$$

$$\Delta \vec{P}^{(2)} = \left[\Delta(\rho\vec{u})^{(12)} + \Delta(\rho\vec{u})^{(2)}\right] V_{im}^a, \tag{10.49b}$$

$$\Delta E^{(2)} = \left[\Delta(\rho e)^{(12)} + \Delta(\rho e)^{(2)}\right] V_{im}^a, \tag{10.49c}$$

represent the sum of the partial-donor [upper index (12)] and the second-order [upper index (2)] contributions; $0 \le \lambda \le 1$ is the limiting factor associated with the given face im, which has been introduced in the RALEF code (November 2011) to suppress possible negative numerical viscosity and too high mass advection. The contributions from the partial-donor treatment are defined by

$$\Delta\rho^{(12)} = \frac{1}{2}\,\mathrm{sign}(V_{im}^a)\,(1-f_d)\,(\rho_i - \rho_{i_o}), \tag{10.50a}$$

$$\Delta(\rho\vec{u})^{(12)} = \frac{1}{2}\,\mathrm{sign}(V_{im}^a)\,(1-f_d)\,(\rho_i\vec{u}_i - \rho_{i_o}\vec{u}_{i_o}), \tag{10.50b}$$

$$\Delta(\rho e)^{(12)} = \frac{1}{2}\,\mathrm{sign}(V_{im}^a)\,(1-f_d)\,(\rho_i e_i - \rho_{i_o} e_{i_o}). \tag{10.50c}$$

The second-order contribution to $\Delta m^{(2)}$ in Eq. (10.49a) is defined by

$$\Delta\rho^{(2)} = \frac{1}{2}\left[\delta\rho_i^{(2)} + \delta\rho_{i_o}^{(2)}\right] - \frac{f_d}{2}\,\mathrm{sign}(V_{im}^a)\left[\delta\rho_i^{(2)} - \delta\rho_{i_o}^{(2)}\right], \tag{10.51}$$

where

$$\delta\rho_i^{(2)} = \alpha_i\,\vec{g}_{\rho i}\cdot(\vec{x}_{im} - \vec{x}_{c,i}), \tag{10.52a}$$

$$\delta\rho_{i_o}^{(2)} = \alpha_{i_o}\,\vec{g}_{\rho i_o}\cdot(\vec{x}_{im} - \vec{x}_{c,i_o}), \tag{10.52b}$$

$\vec{g}_{\rho i}$ and $\vec{g}_{\rho i_o}$ are, respectively, the cell-centered gradients of density ρ in cells i and i_o, $\vec{x}_{im}$ is the center position of face im, $\vec{x}_{c,i}$ and $\vec{x}_{c,i}$ are the center positions of cells i and i_o. For the fractional interpolation lengths α_i and α_{i_o} the following expressions are used in the RALEF code

$$\alpha_i = \max\left\{0,\,\min\left\{1,\,1+\frac{V_{im}^a}{V_i}\right\}\right\}, \tag{10.53a}$$

$$\alpha_{i_o} = \max\left\{0,\,\min\left\{1,\,1-\frac{V_{im}^a}{V_{i_o}}\right\}\right\}, \tag{10.53b}$$

which differ from the original CAVEAT formulae

$$\alpha_{CAV,i} = 1 - \left[\frac{1}{2} - \frac{f_d}{2}\,\mathrm{sign}(V^a_{im})\right]\frac{|V^a_{im}|}{V_i}, \tag{10.54a}$$

$$\alpha_{CAV,i_o} = 1 - \left[\frac{1}{2} + \frac{f_d}{2}\,\mathrm{sign}(V^a_{im})\right]\frac{|V^a_{im}|}{V_{i_o}}. \tag{10.54b}$$

Since for $f_d = 1$ Eqs. (10.53) and (10.54) yield the same result, the formal difference between them can hardly be of any practical importance because it does not make much sense to combine $f_d < 1$ with the second-order scheme. Nevertheless, for $f_d = 0$ Eq. (10.53) gives more bias to the donor cell than (10.54), which appears physically more reasonable. Note also that Eq. (10.53) does not allow non-physical values of the interpolated quantity even for a weaker regular van Leer gradient limiting with `limgrad` $= 1$ and 2 by constraining the interpolation region within the corresponding cell. Expressions for $\Delta(\rho\vec{u})^{(2)}$ and $\Delta(\rho e)^{(2)}$ are fully analogous to Eqs. (10.51) and (10.52) with the only difference that the gradients of the corresponding volume-specific quantity (either $\rho\vec{u}$ or ρe) are used.

The limiting factor λ in Eqs. (10.47) is calculated in two steps, and, respectively, its value is controlled by two user defined parameters $\varepsilon_{a2m} \geq 0$ and $\varepsilon_{a2e} \geq 0$. At the first step the limit λ_m is calculated by constraining mass depletion in the donor cell due to the $\Delta m^{(2)}$ term. More specifically, the value of λ_m is chosen such as to ensure that the $\lambda_m \Delta m^{(2)}$ term carries out no more mass from the donor cell than a fraction ε_{a2m} of what is left after the term $|\Delta m^{(1)}|$ is subtracted, i.e. no more mass than $\varepsilon_{a2m}\,(V_{i_d} - |V^a_{im}|)$, which yields

$$\lambda_m = \begin{cases} \max\left\{0,\, \min\left\{1,\, \varepsilon_{a2m}\dfrac{\rho_{i_d}}{\Delta\rho}\left(\dfrac{V_{i_d}}{|V^a_{im}|} - 1\right)\right\}\right\}, & \Delta\rho > 0 \text{ and } 0 \leq \varepsilon_{a2m} < 1, \\ 1, & \Delta\rho \leq 0 \text{ or } \varepsilon_{a2m} \geq 1, \end{cases} \tag{10.55}$$

where

$$\Delta\rho = \Delta\rho^{(12)} + \Delta\rho^{(2)}. \tag{10.56}$$

Note that the user can easily turn off the limiting due to mass depletion by choosing $\varepsilon_{a2m} \geq 1$. Typically, values $\varepsilon_{a2m} = 0.2$–0.5 would be used.

At the second step the limit λ_e constraining the amount of negative advective dissipation $d\tilde{\epsilon}_a$ of the kinetic energy is evaluated as

$$\lambda_e = \begin{cases} -\varepsilon_{a2e}\dfrac{\max\{\frac{1}{2}\tilde{u}_i^2,\, \frac{1}{2}\tilde{u}_{i_o}^2,\}}{d\tilde{\epsilon}_a}, & d\tilde{\epsilon}_a < 0 \text{ and } 0 \leq \varepsilon_{a2e} < 1, \\ 1, & d\tilde{\epsilon}_a \geq 0 \text{ or } \varepsilon_{a2e} \geq 1, \end{cases} \tag{10.57}$$

where

$$d\tilde{\epsilon}_a = \max\left\{\tilde{\epsilon}_i - \tilde{\epsilon}_{0,i},\, \tilde{\epsilon}_{i_o} - \tilde{\epsilon}_{0,i_o}\right\}. \tag{10.58}$$

Again, the user can easily turn off this limiting step by choosing $\varepsilon_{a2e} \geq 1$. The intermediate values of the mass-specific kinetic energy $\frac{1}{2}\tilde{u}^2$ and internal energy $\tilde{\epsilon}$ in cells i and i_o are calculated by using the intermediate advection terms (10.47) with $\lambda = \lambda_m$, namely,

$$\tilde{u}_i^2 = \left(\frac{\vec{P}_i + \Delta\vec{P}}{m_i + \Delta m}\right)^2, \qquad \tilde{u}_{i_o}^2 = \left(\frac{\vec{P}_{i_o} - \Delta\vec{P}}{m_{i_o} - \Delta m}\right)^2, \tag{10.59}$$

$$\tilde{\epsilon}_i = \frac{E_i + \Delta E}{m_i + \Delta m} - \frac{1}{2}\tilde{u}_i^2, \qquad \tilde{\epsilon}_{i_o} = \frac{E_{i_o} - \Delta E}{m_{i_o} - \Delta m} - \frac{1}{2}\tilde{u}_{i_o}^2. \tag{10.60}$$

The reference dissipationless values $\tilde{\epsilon}_0$ of the mass-specific internal energy in cells i and i_o are given by

$$\tilde{\epsilon}_{0,i_d} = \frac{E_{i_d}}{m_{i_d}} - \frac{1}{2}\left(\frac{\vec{P}_{i_d}}{m_{i_d}}\right)^2, \tag{10.61}$$

$$\tilde{\epsilon}_{0,i_a} = \frac{E_{i_a} - \vec{P}_{ia}^{\,2}/2m_{i_a} + |\Delta m|\,\tilde{\epsilon}_{0,i_d}}{m_{i_a} + |\Delta m|}, \tag{10.62}$$

where again Δm is calculated from Eq. (10.47a) with $\lambda = \lambda_m$. In the end, the minimum

$$\lambda = \min\{\lambda_m,\, \lambda_e\} \tag{10.63}$$

of the two λ-values is used to calculate the final advection terms (10.47).

Practical recommendations: It is recommended to always start with the default values $f_d \equiv$ `fdonor` $= 1.0$, `limgrad` $= 3$, $\varepsilon_{a2m} = 0.5$, and $\varepsilon_{a2e} = 2.0$ when the second-order advection scheme is used. If this option fails with the message of a negative cell mass after advection and the mesh is not too strongly distorted, one can try a smaller value of $\varepsilon_{a2m} \approx 0.1$. If the default option fails because of persistent negative advective dissipation, which sometimes develops into a sort of numerical instability and leads to negative temperatures, one should set a sufficiently small value of $\varepsilon_{a2e} \ll 1$; numerical tests have demonstrated that $\varepsilon_{a2e} = 10^{-3}$ usually fixes the problem. As a final resort, one can set $\varepsilon_{a2e} = 0$.

Correspondence with the code variables:

α_{ALE}	= `alecoef`	ALE coefficient in range $0 \le \alpha_{ALE} \le 1$;
Δt	= `dthydro`	time increment in the current hydrocycle;
$0 \le f_d \le 1$	= `fdonor`	user-defined parameter which defines the amount of donor-cell treatment by advection;
$\varepsilon_{a2m} \ge 0$	= `adv2lim`	user-defined scaling factor for limiting the second-order advection to prevent excessive mass depletion in the donor cell;
$\varepsilon_{a2e} \ge 0$	= `epsie2adv`	user-defined scaling factor for limiting the second-order advection to suppress negative numerical dissipation of the kinetic energy;
$\vec{u}_i$	= `um(i,m), m=1,2`	mesh node velocity;
$\vec{x}_i^{\,L,n+1}$	= `xv(i,m)`	new Lagrangian coordinates of mesh vertices at $(n+1)$-th hydro cycle;
$m_i^{\,L,n+1} = m_i^n$	= `cm(i)`	mesh-cell masses after the Lagrangian phase of the $(n+1)$-th hydro cycle;
$\vec{u}_{c,i}^{\,L,n+1}$	= `ucl(i)`	cell-centered fluid velocities after the Lagrangian phase of the $(n+1)$-th hydro cycle;
$e_i^{\,L,n+1}$	= `tel(i)`	mass-specific total fluid energies after the Lagrangian phase of the $(n+1)$-th hydro cycle;
w_i	= `wt(i)`	weight coefficients in the Winslow rezoning algorithm; cell-centered, I-numbered;
$\vec{x}_i^{\,0,n+1}$	= `xn(i,m)`	initial configuration of rezoned mesh at $(n+1)$-th hydro cycle after the `REZINIT` subroutine;
$\vec{x}_i^{\,1,n+1}$	= `xn(i,m)`	initial configuration of rezoned mesh after special smoothing of selected block boundaries in subroutines `REZSMOO` and `REZSTRE`;
$\vec{x}_i^{\,w,n+1}$	= `xn(i,m)`	almost final configuration of rezoned mesh after `itrezn` rezoning iterations with the `REZITER` subroutine;

$\vec{u}_{r,i}$	= `umr(i,m)=um(i,m)`	mesh rezoning velocities; vertex-centered, I-numbered;
a_{mdx}	= `amaxdx`	parameter defined in subroutine `REZLIM`, which controls the relative coordinate shift from the Lagrangian to rezoned mesh in the boundary smoothing algorithm; default value is 0.7.
c_{sm}	= `crezsm`	coefficient controlling the amplitude of coordinate shift in the boundary smoothing algorithm;
p_{sm}	= `prezsm`	parameter controlling application of the boundary smoothing algorithm;
θ_0	= `arezsm`	sensitivity threshold for turn angles in criterion 1 for application of the boundary smoothing algorithm;
n_{bay}	= `nrezbay`	vertex-number span when probing block edges for the presence of a "vacuum" pocket;
θ_*	= `arezbay`	threshold value for $\Delta\theta^{\pm}_{m,i}$ to apply the pocket-fill algorithm;
c_{bay}	= `crezbay`	amplitude of the "pocket-fill" coordinate shift;

11. PRACTICAL RECOMMENDATIONS FOR REZONING CONTROL

11.1. General remarks

There are many parameters that provide control over different aspects of the mesh rezoning and remapping algorithms. Some of them have been inherited from the CAVEAT code, others have been added in the RALEF package. The values of the most of these parameters can be adjusted via the `namelist/input/`.

The standard way to influence how the rezoning algorithm tends to construct a new mesh is by choosing an appropriate weight function `wt(i)`, calculated in the `subroutine WEIGHT`, file `'f05_remap.f'`. Usually, a new problem-specific formula for the rezoning weight function can be easily programmed by editing Step 2 in this subroutine.

11.2. A nearly Eulerian simulation

Sometimes it may be desirable to suppress the rezoning procedure so that the new mesh $\vec{x}_i^{n+1}$ could be forced to remain arbitrarily close the old one $\vec{x}_i^n$. In particular, this might be an efficient practical option when all the physical boundaries are defined as fixed in space (i.e. as reflective, inflow/outflow or free-slip with `ibc` = 1, 3, 4, or 8) and there are no Lagrangian material interfaces: it often helps to significantly suppress the numerical diffusion near sharp density contrasts in regions with negligible physical motion.

The amplitude $\max(|\vec{x}_i^{n+1}-\vec{x}_i^n|)$ of the mesh displacement can be reduced to an arbitrarily small value

- by setting the value of $\alpha_{ALE} \ll 1$ (say, $\alpha_{ALE} = 10^{-3}$) to ensure practically 100% retraction to $\vec{x}_i^n$ when calculating the initial state $\vec{x}_i^{0,n+1}$ for the rezoning procedure combined with
- a sufficiently small (equal, say, to 0.01) value of the amplitude `aBrackbill` of the Brackbill rezoning algorithm.

11.3. Suppression of tangential rezoning along the mesh boundaries

In some cases it may be desirable to suppress tangential rezoning along certain mesh boundaries. This, in fact, may even be necessary to prevent the mesh being strongly pulled (and finally collapsed there) towards a block corner — which sometimes happens, for example, at an intersection of two outflow boundaries with a strong mass outflow through both of them. Tangential rezoning can be suppressed by setting the flag `ifnotr12bc(ib,iblk)=.true.` for any selected block edge `ib` — preferably the one, along which the mesh is most uniform and the least subject to adaptive changes due to variations of physical quantities (like concentration towards regions with higher matter density).

The effect of setting `ifnotr12bc(ib,iblk)=.true.` will be

- to mark all the physical vertices as fixed in space (i.e. with the flag `fix` set on) along a fixed in space boundary identified with `ibc` = 1, 3, 4, or 8, or
- to mark all the physical vertices as strictly Lagrangian (i.e. with the flag `lvx` set on) along a moving boundary identified with `ibc` = 2, 6, or 9.

Note that any in-line change of the flag `ifnotr12bc(ib,iblk)` requires an appropriate reinitialization of the code because the basic flag array `iflg` has to be reloaded. Therefore, it must be done by editing the `subroutine RUNCTRL` in file '`f10_taskinpt.f`'.

11.4. Parallel translation of the whole mesh in the direction of bulk motion

Another useful opportunity offered by the ALE algorithm is a spatially uniform parallel mesh translation with a prescribed and generally time-dependent translation velocity. This may be a useful option when the region of main interest (the "bulk" of the simulated target) is accelerated in a certain direction and tends to leave the physical volume covered by the computational mesh (either in an Eulerian simulation or in an ALE simulation with fixed in space mesh boundaries).

To activate this option, one has to set the flag `ifmeshslide=.true.`. After that, every time when the `subroutine REZONE` is called, the mesh is uniformly translated with the 2D velocity $\vec{u}_{tr}$ = `(bulk_ux,bulk_uy)`. The components `bulk_ux` and `bulk_uy` of the "bulk" velocity are calculated in the `subroutine BULKVALS`, file '`f05_remap.f`'. If the user wants to change the algorithm for calculation of the mesh translation velocity, he has to edit either steps 0 and/or 6 in the `subroutine REZONE` (file '`f05_remap.f`'), or the `subroutine BULKVALS`, or both.

Here, however, caution is advised: the user must ascertain that the mesh translation velocity is not too high, so that the outflow velocity across a translated outflow boundary does not in fact become an inflow velocity.

12. MATERIAL PROPERTIES

Different parts of every mesh block can be filled with different materials. Any mesh cell can contain only one type of material. On the total, up to 30 different materials are foreseen in the RALEF code, whose properties are listed in a rank-2 array `PROPRTY(i,imat)` = `PROPRTY(1:100,1:30)`. The second index `imat` = $1, 2, \ldots, 30$ of this array is the sequential material number, while its first index `i` = $1, 2, \ldots, 100$ is used to list all the user defined parameters for a given material `imat`. The meaning of the individual elements of the `PROPRTY(i,imat)` array are explained in Table V.

TABLE V: Array `PROPRTY(i,imat)` of material properties

EOS #	1	2	3	5	7	8	
type	polytropic	linear	quadratic	HOM	GLT	SESAME	default
i=1	1.0	2.0	3.0	5.0	7.0	8.0	1.0
2	ρ_{00}	ρ_{00}	ρ_{00}	ρ_{00}	ρ_{00}	ρ_{00}	1.0
3	p_{floor}	p_{floor}	p_{floor}	p_{floor}	p_{floor}	p_{floor}	$-$`CEILING`
4	$A_s = (\gamma+1)/2$	A_s	A_s	A_s	A_s	A_s	4/3
5	γ	c_1	c_1	C	GLT $\#m$	SESAME #	5/3
6	c_V	c_2	c_2	S	Maxwl flag	RSCALE	0.0
7	c_p	c_3	c_3	V_{sw}		ESHIFT	0.0
8		c_4	c_4	C_1		A_1	0.0
9		c_V	c_5	S_1		A_2	0.0
10			c_6	F		A_3	0.0
11			c_7	G		IREVFLG	0.0
12			c_8	H			0.0
13			c_V	I			0.0
14				J			0.0
...				...			
19	e_{floor}	e_{floor}	e_{floor}	e_{floor}	e_{floor}	e_{floor}	`FLOOR`
...				...			
27	A_{mol}	A_{mol}	A_{mol}	A_{mol}	A_{mol}	A_{mol}	0.0
28	Z_{mol}	Z_{mol}	Z_{mol}	Z_{mol}	Z_{mol}	Z_{mol}	0.0
29	z_i (ioniz.deg)	z_i	z_i				0.0
parameters for calculating the conduction coefficient κ and the limiting flux h_l							
model type	$\kappa = \kappa_0 T^n$ $h_l = f_{inh}\rho T^{3/2}$	ad hoc analytic	Spitzer $f_{inh}\frac{\rho z_i T^{3/2}}{A}$	Basko metal	GLT $f_{inh}\frac{\rho z_i T^{3/2}}{A_{mol}}$		
30	1.0	2.0	3.0	5.0	7.0		1.0
31	κ_0		A		GLT $\#m$		0.0
32	n		$(\ln\Lambda)_{min}$				0.0
...				...			
37	f_{inh}	f_{inh}	f_{inh}	f_{inh}	f_{inh}		$-$`CEILING`
...				...			
parameters for calculating radiation opacities							
model type	$k_R = k_{R,0}\rho^\alpha T^\beta$ $k_P = k_{P,0}\rho^\alpha T^\beta$	ad hoc analytic	Kramers free-free		GLT		
40	1.0	2.0	3.0		7.0		1.0
41	$k_{R,0}$	$k_{R,0}$			GLT $\#m$		1.0
42	$k_{P,0}$	$k_{P,0}$					1.0
43	α	T_0					0.0
44	β						0.0
45							0.0
parameters for laser-light opacity							
model type	$k_{las} = k_0\rho^\alpha T^\beta$	ad hoc analytic	Kramers free-free		GLT		
46	1.0	2.0	3.0		7.0		1.0
47	k_0	k_0	$z_{ii,min}$		GLT $\#m$		1.0
48	α	T_0	$z_{ie,min}$				0.0
49	β						0.0
...				...			

13. TIME STEP LIMITATION

Under thermal processes we understand all the heating-cooling terms on the right-hand of the energy equation other than the $p\,dV$ work. In our case we have three such terms due, respectively, to thermal conduction (W_i^T), radiation transport (W_i^r), and possible external energy deposition (W_i^{dep}); see Eq. (6.1) in the RALEF-2D report [**?**]. Because time discretization of these thermal terms is done by using the symmetric semi-implicit (SSI) method, we need an additional [with respect to the usual Courant-Friedrichs-Lewy (CFL) condition] "thermal" constraint on the value of the time step Δt. This constraint is based on the requirement that the temperature increment $|\tilde{T}_i - T_i|$ in cell i at the SSI phase of the Lagrangian step must not be too large, namely, on the condition

$$\left|\tilde{T}_i - T_i\right| = \left|\frac{W_i\Delta t + \delta_i}{c_{V,i}M_i + D_i\Delta t}\right| \le \varepsilon_0(T_i + T_s), \tag{13.64}$$

where $W_i = W_i^{dep} + W_i^T + W_i^r$ is the total thermal heating power of cell i, $c_{V,i}M_i$ its heat capacity, $D_i = -\partial W_i/\partial t$, $\delta_i = \delta_{T,i} + \delta_{r,i}$ is the SSI energy correction, taken from the previous hydro cycle, and ε_0 and T_s are two user-defined free parameters.

Presently there are two versions of the thermal limit on Δt implemented in the RALEF-2D code: a "hard" one and a "soft" one. They are distinguished by the value of the user-defined parameter $c_{1,tst}$: for $c_{1,tst} = 0$ the "hard" version is active, for $c_{1,tst} > 0$ the "soft" one applies. The "hard" version of the time-step limit corresponds to the most strict implementation of condition (13.64). Here, however, the main obstacle is the fact that, when δ_i is calculated, the value of Δt for the next hydro cycle (where δ_i must be redeposited) is not known. As a consequence, the "hard" time-step limit is realized by splitting the criterion (13.64) into the following two conditions

$$\left|\frac{W_i\Delta t}{c_{V,ij}M_{ij} + \Delta t D_i}\right| \le (\varepsilon_0 - \varepsilon_1)\,(T_i + T_s)\,, \quad \left|\frac{\bar{\delta}_i}{c_{V,i}M_i}\right| \le \varepsilon_1\,(T_i + T_s)\,, \tag{13.65}$$

where ε_1 is an additional free parameter in our criterion. Clearly, one must ensure that $\varepsilon_1 < \varepsilon_0$. The bar over δ_i in Eq. (13.65) means that this is a "postponed" quantity, to be used for calculating $\tilde{T}_i - T_i$ only in the next hydro cycle. Because $\bar{\delta}_i$ is, in its turn, proportional to the current value of Δt (see Eq. (5.36) in Ref. [2] and Eq. (?) in Ref. [**?**]), we can, by choosing sufficiently small values of $\varepsilon_0 > \varepsilon_1$ and T_s, always keep relative temperature variation $|\tilde{T}_i - T_i|/T_i$ at one time step within desired limits.

The principal drawback of the "hard" limit (13.65) is that in many practical situations, where one has $D_i\Delta t \gg 1$, this constraint turns out to be too restrictive and either unnecessarily slows down the simulation by a significant factor or completely blocks it. Typically it occurs when strong heat-conduction fluxes are present in a tenuous medium with relatively small values of cell heat capacities $c_{V,i}M_i$. To speed up the simulation in such cases, an alternative "soft" version of the criterion (13.64) has been implemented.

The "soft" time-step control is based on the following strategy: we keep track of the relative temperature change

$$\frac{\delta T(\Delta t)}{T} = \max_i \left\{(T_i + T_s)^{-1}\left|\frac{W_i\Delta t + \delta_i}{c_{V,i}M_i + D_i\Delta t}\right|\right\} \tag{13.66}$$

and make small corrections to Δt in order to keep $\delta T/T$ within a range $0.5\varepsilon_0 < \delta T/T < 0.6\varepsilon_0$. More precisely, the SSI phase of every hydro cycle starts with a trial value of the time step

$$\Delta t_* = \min\left\{c_{dtgr}\Delta t_{prev};\, \Delta t_{CFL};\, \Delta_{ev}\right\}, \tag{13.67}$$

where $c_{dtgr} > 1$ is a user-defined growth factor (typically, $c_{dtgr} = 1.05$–1.1), Δt_{prev} is the value of Δt in the previous hydro cycle, Δt_{CFL} is the value of Δt obtained from the purely hydrodynamic CFL criterion, and Δt_{ev} is some other eventual time step limit. Next, the maximum relative temperature change $\delta T(\Delta t_*)/T$ is calculated for the starting value Δt_*, which is subsequently modified to (a "fine-tuning" correction)

$$\Delta t = \begin{cases} \Delta t_*, & \delta T(\Delta t_*)/T \le 0.5\varepsilon_0, \\ \min\{\Delta t_{prev}, \Delta t_*\}, & 0.5\varepsilon_0 < \delta T(\Delta t_*)/T \le 0.6\varepsilon_0, \\ \dfrac{\min\{\Delta t_{prev}, \Delta t_*\}}{1 + c_{1,tst}(c_{dtgr} - 1)}, & 0.6\varepsilon_0 < \delta T(\Delta t_*)/T \le 1.2\varepsilon_0. \end{cases} \tag{13.68}$$

This correction is made only once, i.e. the above "fine-tuning" procedure is non-iterative. In Eq. (13.68) one can make use of the values of $c_{1,tst} > 1$ to speed up the reduction of Δt relative to its increase rate by means of the parameter $c_{dtgr} > 1$.

Occasionally the "fine-tuning" procedure (13.68) becomes unstable, and one ends up with a large temperature variation $\delta T(\Delta t_*)/T > 1.2\varepsilon_0$. Such a situation is considered as a failure (or a "crash") of the "soft" time-step control procedure. Here one has to impose a more dramatic reduction of Δt to bring down the relative temperature variation $\delta T/T$, and it has to be done in an iterative loop. The problem is that, if we want to conserve energy, we are not allowed to change δ_i as Δt is reduced. As a practical solution, we choose to sacrifice energy conservation and scale down the values of δ_i (in proportion to Δt) in those cells i where $(\delta T/T)_i > 0.5\varepsilon_0$. In other words, when the "fine-tuning" procedure (13.68) crashes, we violate strict energy conservation and bring down the temperature change to a level $\delta T/T < 0.5\varepsilon_0$ by strong reduction of Δt in an iterative loop similar to that in the "hard" version of the thermal time-step control algorithm. If $D_i \Delta t \gg 1$, the "crash" reduction of Δt may be by several orders of magnitude.

For optimal performance, the following values of the user-defined control parameters can be recommended:

- for the "hard" option of the thermal time-step control:

$$c_{1,tst} = 0, \quad \varepsilon_0 = 0.1\text{–}0.2, \quad \varepsilon_1 = 0.5\varepsilon_0; \tag{13.69}$$

- for the "soft" option of the thermal time-step control:

$$c_{1,tst} = 1\text{–}2, \quad \varepsilon_0 = 0.06. \tag{13.70}$$

In the"soft" option, the value of ε_1 is irrelevant. The values of c_{dtgr} are allowed in the range $1 < c_{dtgr} < 2$, recommended are $c_{dtgr} = 1.05$–1.1. In simulations of radiative hohlraums, the "soft" option of thermal time-step control allowed to reduce the computing time by about a factor 2–3.

Correspondence with the code variables:

$c_{1,tst}$ = `c1dtssi` – user-defined parameter which controls application of a "hard" or a "soft" thermal time-step limit; default = 0;

c_{dtgr} = `dtgrow` – user-defined growth factor for increasing the time step Δt; default = 1.05;

ε_0 = `eps0ssi` – principal time-step control parameter at the SSI stage; default = 0.1;
ε_1 = `eps1ssi` – secondary time-step control parameter at the SSI stage; default = 0.05;
T_s = `tempsns` – sensitivity threshold for T variation;
T_{flr} = `tempflr` – absolute minimum for T values; default = `floor`;
$\delta T/T$ = `dtmpssi` – relative temperature change at the SSI phase of the hydro cycle;
Δt_{prev} = `dtprev` – time step in the previous hydro cycle;

14. PARALLELIZATION WITH OPENMP

Clearly, the computational work needed for calculation of the radiative heating powers W_i^r can be easily divided into independent blocks corresponding to different beamlet directions $\vec{\Omega}_L$ and different photon frequencies $[\nu_k, \nu_{k+1}]$ that can be processed in parallel. And though all the frequency groups $[k] = 1, 2, \ldots, N_\nu$ are independent from one another, this is not the case for angular directions $\vec{\Omega}_L$, which have to be processed in groups.

In the case of Cartesian (x, y) geometry every *independent* angular group combines 4 beamlets from the 4 different octants: these 4 beamlets have the same value of index $l = 1, 2, \ldots, N_\Omega$, but different values of the octant indices $i_{ox} = \pm 1$, $i_{oy} = \pm 1$. Thus, for `IRADIAL` $= 0$ all independent angular groups have the same number of individual beamlets, and there are in total N_Ω independently processable angular groups.

In the axi-symmetric (r, z) geometry the division into independent angular groups is more complex, and different independent groups generally have different number of individual beamlets. To make the distribution of individual beamlets over the independently processable groups more even, we assume that all the angular beamlets can be divided into $N_{\Omega bdls}$ independently processable Ω-*bundles* of beamlets. One bundle may, in fact, contain either one big, or two smaller independently processable groups. Generally, $N_{\Omega bdls} \leq N_\Omega$. For `IRADIAL` $= 0$ we always have $N_{\Omega bdls} = N_\Omega$.

To optimize work distribution among N_{thr} different threads, we combine the $N_{\Omega bdls}$ Ω-bundles and the N_ν frequencies into a single iteration loop, with the iteration index K running through the values $K = 1, 2, \ldots, N_K$; in the simplest case $N_K = N_\nu \times N_{\Omega bdls}$. Thus, the principal parallelized section of the RALEF-2D code is represented by a combined frequency×direction do-loop

```
!$OMP DO SCHEDULE(STATIC,nchunk(kOMPass))

    do K=1,1,nKloop(kOMPass)


     ...

    enddo

!$OMP END DO
```
(14.71)

In a regular hydrocycle this loop is passed only once, and `kOMPass` $= 1$. In rare hydrocycles, where the spectral diagnostics is to be computed and written out, this loop is passed twice: first time with `kOMPass` $= 1$ for the main radiation-hydro calculation, and second time with `kOMPass` $= 2$ for the diagnostics.

From the point of view of parallelization logic, the total number $N_K = \texttt{nKloop(kOMPass)}$ of combined iterations is divided into N_{thr} *chunks* of work in a static manner, so that each chunk consists of $N_{ch} = \texttt{nchunk(kOMPass)}$ sequential iterations performed one after another by a single thread; N_{thr} is the total number of threads. Thus, each thread processes one and only one chunk of work; the last chunk is allowed to have less than N_{ch} iterations.

From the point of view of physics, the total number N_K of combined iterations consists of $N_{\Omega bdls}$ separate *frequency clusters* because for each Ω-bundle the angular information is calculated only once, and then is used N_ν times by each of the N_ν frequencies. Evidently, the angular information must be calculated anew, firstly, at the start of every chunk and, secondly, by transition from one frequency cluster to another. Because one does not know a priori where in a chunk such transition will occur, it is convenient to emulate calculation of angular information by any such transition as an extra frequency group with a number $[k] = 0$. The latter is justified by the fact that in our case the CPU time required to recalculate the angular information is approximately equal to the CPU time needed to process one frequency. Thus, in iterations with $[k] = 0$ the index $i_{\Omega bdl}$ of the Ω-bundle is incremented by 1, the angular information is calculated anew, and no frequency is processed. Then, in subsequent iterations with $[k] = 1, 2, \ldots$ no angular information is calculated, and one frequency group $[k] \geq 1$ is processed at a time.

TABLE VII: Distribution of work among $N_{thr} = 4$ threads by OpenMP parallelization for the case of $N_{\Omega bdls} = 5$, $N_\nu = 3$, $N_{\nu cl} = 4$, $N_{ch} = 5$.

thread	0					1					2					3				
K	1,	2,	3,	4,	5	6,	7,	8,	9,	10	11,	12,	13,	14,	15	16,	17,	18,	19	
$[k]$	1,	2,	3,	0,	1	2,	3,	0,	1,	2	3,	0,	1,	2,	3	0,	1,	2,	3	
$i_{\Omega bdl}$	1,	1,	1,	2,	2	2,	2,	3,	3,	3	3,	4,	4,	4,	4	5,	5,	5,	5	

TABLE VIII: Distribution of work among $N_{thr} = 4$ threads by OpenMP parallelization for the case of $N_{\Omega bdls} = 4$, $N_{\nu cl} = N_\nu = 4$, $N_{ch} = 4$.

thread	0				1				2				3			
K	1,	2,	3,	4	5,	6,	7,	8	9,	10,	11,	12	13,	14,	15,	16
$[k]$	1,	2,	3,	4	1,	2,	3,	4	1,	2,	3,	4	1,	2,	3,	4
$i_{\Omega bdl}$	1,	1,	1,	1	2,	2,	2,	2	3,	3,	3,	3	4,	4,	4,	4

TABLE IX: Distribution of work among $N_{thr} = 4$ threads by OpenMP parallelization for the case of $N_{\Omega bdls} = 3$, $N_\nu = 6$, $N_{\nu cl} = 7$, $N_{ch} = 5$.

thread	0					1					2					3				
K	1,	2,	3,	4,	5	6,	7,	8,	9,	10	11,	12,	13,	14,	15	16,	17,	18,	19	20
$[k]$	1,	2,	3,	4,	5	6,	0,	1,	2,	3	4,	5,	6,	0,	1	2,	3,	4,	5	6
$i_{\Omega bdl}$	1,	1,	1,	1,	1	1,	2,	2,	2,	2	2,	2,	2,	3,	3	3,	3,	3,	3	3

Finally, we arrive at the following values of the parameters governing the parallelization process. The

$$N_K = \begin{cases} N_{\Omega bdls} \times N_{\nu cl}, & N_{\nu cl} = N_\nu, \\ (N_{\Omega bdls} \times N_{\nu cl}) - 1, & N_{\nu cl} = N_\nu + 1, \end{cases} \tag{14.72}$$

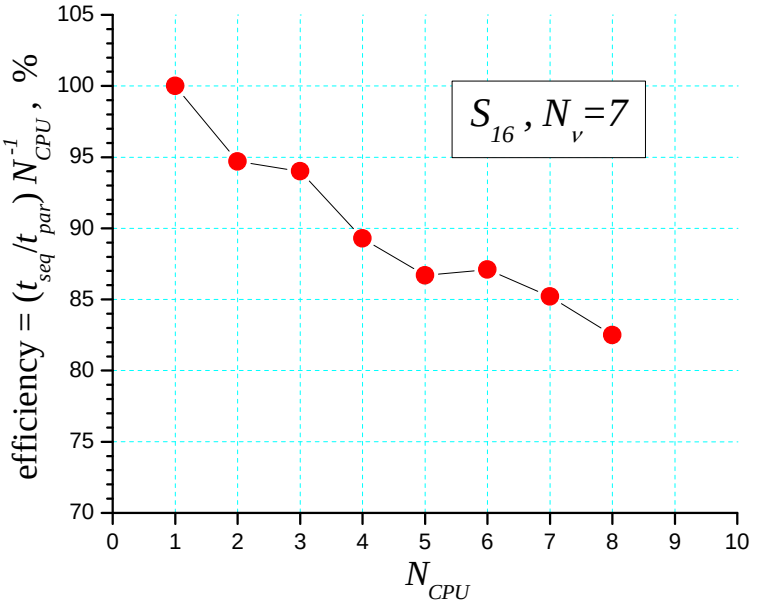


FIG. 14.7: Relative efficiency (in percentage points) of the present parallelization scheme, obtained for $N_{\Omega bdls} = 36$, $N_\nu = 7$ with $N_{thr} \leq 8$ threads.

combined frequency×direction iterations consist of $N_{\Omega bdls}$ frequency clusters; every frequency cluster has

$$N_{\nu cl} = \begin{cases} N_\nu, & N_{thr} \text{ is a multiple of } N_{\Omega bdls}, \text{ or } N_{\Omega bdls} \text{ is a multiple of } N_{thr}, \\ N_\nu + 1, & \text{otherwise}, \end{cases} \tag{14.73}$$

frequencies (or frequency-equivalent elements); the N_K combined iterations are divided into N_{thr} chunks of work, each chunk comprising

$$N_{ch} = \begin{cases} \mathrm{INT}\,(N_K/N_{thr})\,, & N_K = N_{thr} \times \mathrm{INT}\,(N_K/N_{thr})\,, \\ \mathrm{INT}\,(N_K/N_{thr}) + 1, & N_K > N_{thr} \times \mathrm{INT}\,(N_K/N_{thr}) \end{cases} \tag{14.74}$$

successive iterations to be performed by one parallel thread. The current index of the Ω-bundle can be calculated as

$$i_{\Omega bdl} = \begin{cases} \mathrm{INT}\,[(K-1)/N_{\nu cl}] + 1, & N_{\nu cl} = N_\nu, \\ \mathrm{INT}\,(K/N_{\nu cl}) + 1, & N_{\nu cl} = N_\nu + 1. \end{cases} \tag{14.75}$$

The described parallelization scheme is illustrated on three characteristic examples in Tables VII–IX. Figure 14.7 shows the efficiency of parallel simulation for $N_{\Omega bdls} = 36$, $N_\nu = 7$ and $N_{thr} = 1, 2, \ldots, 8$.

Correspondence with the code variables:

$N_{\Omega bdls}$	= `nOmbndls`	number of Ω-bundles;
$i_{\Omega bdl}$	= `iOmbndl`	sequential number of the current Ω-bundle;
$N_{\nu cl}$	= `nnuclust(kOMPass)`	number of frequencies (frequency-element positions) in a frequency cluster;
N_{ch}	= `nchunk(kOMPass)`	number of combined frequency × direction iterations in one chunk;
N_K	= `nKloop(kOMPass)`	total number of combined frequency × direction iterations;

15. RAM REQUIREMENTS

Principal RAM requirements for simulations with RALEF-2D are associated with the field arrays (for such variables like fluid density, flow velocity, etc.) allocated in `module COMDECK1`. The basic dimension unit for these arrays is given by the parameter `msize`, which represents the dimension of an array for a scalar physical variable on the entire 2D mesh. Here we use the memory block needed for a single scalar `real(8)` array of size `msize` as a convenient unit of RAM and call it an *r8-block*. As an example of an advanced high-resolution simulation, we may consider the case of `msize` = 1 000 000, where the r8-block is equal to 8 MB of RAM.

Module `COMDECK1` contains two types of field arrays:

- "sovereign" arrays are once and for all associated with the corresponding physical variables (like temperature `temp(msize)`, for example);
- working arrays (like `w1w01(msize), w2w01(2*msize)` and so on) can be used for different physical variables in different parts of the code; association with physical variables is done by using corresponding pointers.

There is a clear naming convention for the working arrays.

The RAM requirements for the code version **RAD-XY-2010.11** are as follows

- 34 basic r8-blocks for "sovereign" arrays needed for hydrodynamics and thermal conduction;
- 6 additional r8-blocks for "sovereign" arrays needed for radiation transport and laser energy deposition;
- 38 basic r8-blocks for working arrays needed for hydrodynamics and thermal conduction;
- 32 additional r8-blocks for working arrays needed for radiation transport and laser energy deposition;

Thus, the total RAM requirement is 110 r8-blocks, which amounts to about 1 GB of RAM for a high-resolution simulation. However, such an estimate will only be correct for a sequential simulation without parallelization. If an OpenMP version of the code is used with N_{thr} parallel threads, then the actual RAM requirement will be

$$110 + 56(N_{thr} - 1) \tag{15.76}$$

r8-blocks, because 56 r8-blocks must be allocated to thread-private working arrays. Then, a high-resolution simulation with 8 threads will require about 4 GB of RAM.

From the above data one infers also that in simulations without radiation transport and laser deposition one can save about 30% of RAM by commenting out the corresponding memory allocation statements in `module COMDECK1` for these processes, and by using the dummy versions of the files `"f09_rad.f"` and `"f11_taskdepo.f"`.

16. INSTRUCTIONS FOR SETTING UP A NEW PROBLEM

In general, to simulate a new problem with the RALEF-2D code, one has to perform the following three groups of operations:

1) provide a database for tabular EOS and opacities (if needed);

2) assign problem parameters in the input file '`in2d`';

3) edit, compile and link the 13 principal Fortran-90 source-code files, named "`f00_comod.f`", "`f01_main.f`", ..., "`f12_taskout.f`".

16.1. Step 1: prepare the EOS and opacity database

1. Analytical equation of state and/or opacities

If one of the preprogrammed analytical models is used for the equation of state, the thermal conduction and the radiation absorption coefficients, then the parameters of the corresponding analytical model are set up in the input file '`in2d`' at Step 2, described in the next subsection. In this case Step 1 is not needed.

2. GLT equation of state and/or opacities

If either for the equation of state, or for the thermal conduction coefficient, or for the radiation absorption coefficients a tabular option in the form of the general logarithmic tables (GLT) is chosen by setting `proprty(1,imat)` = 7.0 [or `proprty(30,imat)` = 7.0, or `proprty(40,imat)` = 7.0], then the files "`TABLE_GLT-EOS`" and/or "`TABLE_GLT-TCRAD`" with corresponding tables must be generated. This is accomplished by running separate code packages GLT-EOS and GLT-TCR: the GLT-EOS package generates the file "`TABLE_GLT-EOS`", which contains the GLT database for tabular EOS; the GLT-TCR package generates the file "`TABLE_GLT-TCRAD`", which contains the GLT database for tabular thermal conduction coefficient and/or tabular Rosseland, Planckian and spectral opacities. In this way tabular opacities and/or thermal conductivity can be combined with any analytical EOS and vice versa.

GLT equation of state. The GLT-EOS is a package for filling the EOS tables with data, generated by a selected source EOS model. The main program of this package, `program GTESFILL`, is in file "`tabeos_GLT.f`". The routines of every separate source EOS model are collected into separate files, named after the corresponding source model. To the moment, an interface for the following three source EOS models has been implemented

- the "Novik" EOS (file "`sourceos_Novik.f`") is based on the original tables in files "`kesp.*`", produced by the THERMOS code of V.G. Novikov (KIAM, 2008); the "Novik" EOS data are available for the elements with $Z = 1, 6, 13, 22, 29, 74, 79, 102(CH_2)$; the Maxwell construction is not possible;

- the "NVK_2" EOS (file "`sourceos_NVK_2.f90`") is based on the original tables in files "`kesp_2.*`", produced by the THERMOS code of V.G. Novikov and modified by A.S. Grushin to include the cold curve, the realistic behavior near the normal state, and the Maxwell construction (if requested); presently available for $Z = 4, 6, 13, 29, 79, 83, 102(CH_2)$.

- the "Basko" EOS (file "`sourceos_Basko.f`") is based on the original average-atom model by M.M. Basko; presently available for $Z = 1, 2, 3, 4, 5, 6, 7, 8, 11, 13, 18, 22, 26, 29, 42, 47, 54, 55, 74, 79, 82, 83, 92$; Maxwell construction is not possible.

As a result, to generate a GLT-table file "TABLE_GLT-EOS", one has to compile a FORTRAN package consisting of the following 6 files: "f00_comod.f", "tabeos_GLT.f", "tabdbase.f", "sourceos_Novik.f", "sourceos_NVK_2.f90", and "sourceos_Basko.f". More precisely, in whole one has to go through the following steps:

1. For the "Novik" (or "NVK_2") EOS do the following: copy the corresponding original Novikov files "kesp.*" (or "kesp_2.*") for all the selected materials into the subdirectory "tabnvk", i.e. provide all the needed source files "tabnvk/kesp.*" (or "tabnvk/kesp_2.*").

2. Set the GLT-table parameters by editing Step 2 in program GTESFILL (file "tabeos_GLT.f").

3. Check whether parameter NNTBLGTES in module COMGLTAB (file "f00_comod.f") is not smaller than

$$\sum_m [\texttt{nrgtes}(m)+1][4+8(\texttt{negtes}(m)+1)], \quad m=1,\texttt{nmatgtes};$$

   if no check is done and NNTBLGTES is too small, the user will later be prompted to increase NNTBLGTES accordingly.

4. Modify (if needed) the value of RASCEOS = 1.2 in the subroutine SRCEOS, file "sourceos_*.f*".

5. Compile the package and run GTESFILL.

For the "Basko" EOS step 1. is not needed.

Example: a tabular EOS for two materials, Au and C, set up by using the "NVK_2" and the "Basko" source EOS models; table dimensions are 10^{-8} g/cc $\leq \rho \leq 10^2$ g/cc (400 intervals along the $\ln\rho$ axis), 0.1 eV $\leq T \leq$ 1 keV (120 intervals along the $\ln e$ axis); no Maxwell construction; the corresponding table will be generated with the following revision of Step 2 in program GTESFILL:

```
!@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ user-defined:
ulengtes=1.d0
utimgtes=1.d-6
umasgtes=1.d0
utmpgtes=1.60217733d-9

!--------------------------------------
!  Step 2:  Select materials ...
!--------------------------------------

nmatgtes=2

m=1
tabnamees(m)=sceostag_NVK_2
Maxwell(m)=.false.
zmolgtes(m)=79.d0
amolgtes(m)=196.97d0
nrgtes(m)=400
negtes(m)=120
rho0(m)=1.d-8
rho1(m)=1.d2
```

```
temp0(m)=1.d-4
temp1(m)=1.d0

m=2
tabnamees(m)=sceostag_BASKO
Maxwell(m)=.false.
zmolgtes(m)=6.d0
amolgtes(m)=12.d0
nrgtes(m)=400
negtes(m)=120
rho0(m)=1.d-8
rho1(m)=1.d2
temp0(m)=1.d-4
temp1(m)=1.d0
!@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ user-defined.
```

The results of the GLT-EOS table generation are summarized in the output file "GLT-EOS_protocol.dat"; one should pay attention to the accuracy of the tabular EOS: the L_2 errors at table nodes must be on the order of 10^{-7} (the rounding errors for `real(4)` numbers), and about 1% at midpoints of the tabular grid.

GLT conduction coefficient and opacity. The file "TABLE_GLT-TCRAD", containing tables for the thermal conduction coefficient and opacities, is generated by a separate FORTRAN package GLT-TCR. The main program of this package, `program GTCRFILL`, is in file "tabtcr_GLT.f". Similar to the GLT-EOS, the GLT-TCR package can use different source models of the conduction coefficient and opacities. To the moment, an interface for only one source TCR model has been implemented, namely for the "Novik" TCR model (file "sourcetcr_Novik.f"), based on the original tables in files "k0sp.*" and "kesp.*", produced by the THERMOS code of V.G. Novikov (KIAM, 2008); the "Novik" TCR data are available for the elements with $Z = 1, 6, 13, 22, 29, 74, 79, 102(CH_2)$.

As a result, to generate a GLT-table file "TABLE_GLT-TCR", one has to compile a FORTRAN package consisting of the following 4 files: "f00_comod.f", "tabtcr_GLT.f", "tabdbase.f", and "sourcetcr_Novik.f". More precisely, in whole one has to go through the following steps:

1. Copy the corresponding original Novikov files "k0sp.*" and "kesp.*" for all the selected materials into the subdirectory "tabnvk", i.e. provide all the needed source files "tabnvk/k0sp.*" and "tabnvk/kesp.*").

2. Adjust PARAMETER values in `module COMGLTAB` in file "f00_comod.f", and in `module SOURCTCR_Novik_com` in file "sourcetcr_Novik.f" to match the current version of of the Novikov source files; also, update if needed the NVK-table readout parameters in subroutines `INISRCTC_NOVIK` and `INISRCOPA_NOVIK`.

3. Set the GLT-table parameters by editing Steps 2 and 3 in `program GTCRFILL` (file "tabtcr_GLT.f").

4. Compile the package and run `GTCRFILL`.

Example: a table for conduction coefficient and opacities of one material Au; table dimensions for the conduction coefficient are 10^{-6} g/cc $\leq \rho \leq 10^{2}$ g/cc (120 intervals along the $\ln\rho$ axis), 1 eV $\leq T \leq$ 0.39811 keV (90 intervals along the $\ln T$ axis); table

dimensions for opacities are 10^{-6} g/cc $\leq \rho \leq 10^{2}$ g/cc (120 intervals along the $\ln \rho$ axis), 1 eV $\leq T \leq$ 0.39811 keV (60 intervals along the $\ln T$ axis); 5 frequency groups are chosen for hydrodynamics, and 200 frequency groups for diagnostics over the entire frequency range of 1 eV $\leq h\nu \leq$ 10 keV; the corresponding table will be generated with the following revision of Steps 2 and 3 in `program GTCRFILL`:

```
!@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ user-defined:
ulengtes=1.d0
utimgtes=1.d-6
umasgtes=1.d0
utmpgtes=1.60217733d-9

!-------------------------------------------------------
!  Step 2:  Set user-defined parameters of the GLT-TC tables
!  (in the above defined units).
!-------------------------------------------------------
!  Set material and grid parameters:
nmatgtc=1

m=1
tabnametc(m)=sctctag_NOVIK
zmolgtc(m)=79.d0
amolgtc(m)=196.97d0
nrgtc(m)=120
nTgtc(m)=90
rho0c(m)=1.d-6
rho1c(m)=1.d2
temp0c(m)=1.d-3
temp1c(m)=.39811d0

!-------------------------------------------------------
!  Step 3:  Set user-defined parameters of the GLT-RAD tables
!  (in the above defined units).
!-------------------------------------------------------
!  ifnuPlRos(m) = 1 -> weight function = 1 for frequency averaging
!  inside a spectral group
!  (i.e.  when calculating SCKNU in subr SRCKNU);
!  ifnuPlRos(m) = 2 -> weight function = Planckian;
!  ifnuPlRos(m) = 3 -> weight function = Rosseland;
!.......................................................

!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
!  Step 3.1:  Set material and grid parameters.
!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
nmatgtr=1

m=1
tabnametr(m)=scopatag_NOVIK
zmolgtr(m)=79.d0
amolgtr(m)=196.97d0
nrgtr(m)=120
nTgtr(m)=60
rho0r(m)=1.d-6
rho1r(m)=1.d2
temp0r(m)=1.d-3
temp1r(m)=.39811d0
```

```
ifnuPlRos(m)=2
!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
!  Step 3.2:  Set dimensions of frequency sets (allowed are
!  only nfrsetsGLT=1 and nfrsetsGLT=2).
!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
nfrsetsGLT=2
nfreqsGLT(1)=5
nfreqsGLT(2)=200
!@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ user-defined.
...
!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
!  Step 3.3:  Set frequency set # 1.
!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
IF(nfreqsGLT(1).ge.1) THEN
!@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ user-defined:
frqnsGLT(1,1)=1.d-3
frqnsGLT(nfreqsGLT(1)+1,1)=10.d0
if(nfreqsGLT(1).ge.2) then
frqnsGLT(2,1)=.1d0
frqnsGLT(3,1)=.3d0
frqnsGLT(4,1)=1.d0
frqnsGLT(5,1)=3.d0
endif
!@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ user-defined.
ELSE
nfreqsGLT(1)=0
ENDIF
!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
!  Step 3.4:  Set frequency set # 2.
!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
if(nfrsetsGLT.le.1) then
nfreqsGLT(2)=0
goto 390
endif
IF(nfreqsGLT(2).ge.1) THEN
!@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ user-defined:
frqnsGLT(1,2)=1.d-3
frqnsGLT(nfreqsGLT(2)+1,2)=10.d0
if(nfreqsGLT(2).ge.2) then
!  Equally spaced in log(nu) between wwnu1 and wwnun:
wwnu1=frqnsGLT(1,2)
wwnun=frqnsGLT(nfreqsGLT(2)+1,2)
wwdlf=log(wwnun/wwnu1)/dble(nfreqsGLT(2))
do k=2,nfreqsGLT(2)
frqnsGLT(k,2)=wwnu1*exp(wwdlf*(k-1))
enddo
endif
!@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ user-defined.
ELSE
nfreqsGLT(2)=0
ENDIF
```

```
390 continue
```

16.2. Step 2: assign problem parameters in the input file ‘in2d’

The values of the problem parameters included into the namelist/input/ are assigned by editing the text file ‘in2d’. The first line of this file before the operator $INPUT contains the job title. An example of the ‘in2d’ file for simulation of an empty cylindrical hohlraum is given below.

```
Au cyl.hohlraum:  case 01:  empty, 1 nu-group
&input !  Begin NAMELIST "input"
nrestart=0
!  Units of measurement (in terms of CGS units):
unilngth=1.d-1
unitime=1.d-8
unimass=1.d-3
unitemp=1.60217733d-9

!  Geometry and mesh:
igeom =3 !  r-theta polar mesh with iradial=0
iradial=0
nblks=1
nprts(1,1)=3 !  3 parts along theta
ncell(1,1,1)=54,24,180
ncell(1,2,1)=40
xxl(1,1,1)=24.d0, 78.d0, 102.d0, 336.d0 !  degrees
xxl(1,2,1)=0.35d0, .365d0 !  radial dimensions

!  Material properties:
matnum(1,1)=1,1,1
proprty(1,1)=7.d0 !  EOS type
proprty(2,1)=.1d0 !  rho00, used for wt(i) in REZONE
proprty(4,1)=1.2d0 !  strong shock parameter
proprty(5,1)=1.d0 !  m-number in GLT tables
proprty(27,1)=196.97d0, 79.d0
proprty(30,1)=7.d0, 1.d0 !  conduction
proprty(37,1)=-.5d0
proprty(40,1)=7.d0, 1.d0 !  opacity
proprty(46,1)=3.d0 !  laser absorption

!  Initial and boundary conditions:
rho0(1,1)=19.d0, 1.d-4, 19.d0
pr0(1,1)=1.d-7, 1.d-7, 1.d-7
pbc(1,1,1)=1.d-7, 1.d-7, 1.d-7
```

```
pbc(1,2,1)=1.d-7, 1.d-7, 1.d-7
pbc(1,3,1)=1.d-7
pbc(1,4,1)=1.d-7

!  Radiation, thermal treatment:
itemp=4
nfreqsets=2
nfreqs=1,200
kradSn=3
ifshadow=1
iflasdep=.true.
iradfbc=0
iradbbc=0
TEMPFLR=3.d-5
TEMPSNS=2.d-3
EPS0SSI=.2d0
EPS1SSI=.1d0

!  Runtime, printout control:
twfin=.2d0
ncycmax=1
ntty=100
twmovi=0.0d0, 1.d7, .05d0
twprnt=0.d0, 1.d35, .05d0
twfilm=0.d0, 1.d1, 1.3d2, 2.d35
twspctr=0.d0, .1d9, .05d9
twdump=1.d2, 1.d35, 1.d2

!  ALE, rezone control:
iorder=1
alecoef=.99d0
dt0=1.d-6
dtmin=1.d-60
dtmax=1.d-4
dtfac=0.5

nadvskp=1
wtampl=6.d0
nsmooth=4
itrezn=5

ibcrezn(1,1)=2,1,1,1
crezsm=.4d0
prezsm=1.5d0
arezsm=.03d0
nrezbay=12
```

```
arezbay=.6d0
crezbay=.4d0

/ !  End NAMELIST "input"
```

The meaning of all the parameters that can be set via this namelist is explained in the documentation file `"00glossar_RALEF.txt"`.

16.3. Step 3: edit the source-code files

In most cases setting up a new problem requires, besides provision of the input file `'in2d'`, editing certain pieces of the source code, when, for example, a problem-specific heat source, non-standard boundary conditions, a problem-specific output, ... are to be set up. Typically, the code blocks that are to be programmed anew are in the files `f10_taskinpt.f`, `f11_taskdepo.f`, `f12_taskout.f` and marked as

```
!@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ user-defined:
    ...
    ...
!@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ user-defined.
```

Also, because of the explicit-shape mode of memory allocation, inherited from the CAVEAT code, certain memory-allocation parameters have often to be adjusted by editing the `module COMPAR` in file `f00_comod.f`.

1. Memory-allocation and general accuracy parameters

The most important parameter, which controls the amount of RAM allocated to the problem is `msize`. If radiation transport is activated (i.e. for `itemp` ≥ 3), the next important memory-allocation parameter is `kradSnmx`. If memory is tight, the minimum possible values must be assigned to these two parameters.

Less important are the other memory-control parameters `nb`, `ns`, `np`, `mszmbo ntiermx`, `nnblas`, `mszfreq`, which define the sizes of certain arrays. The values of these parameters are set by editing the `module COMPAR` in the source-code file `f00_comod.f`.

2. Laser energy deposition

To activate laser energy deposition, go through the following steps:

- in the `namelist/input/`: set `iflasdep=.true.` to activate laser deposition; set `proprty(46,imat)` $= 3$ to use the analytical inverse-bremsstrahlung absorption coefficient for the laser light;
- set all the parameters that specify the laser beam configuration [number of beams `nblas`, propagation direction `(omeblasx(i),omeblasy(i))` of every beam `i`, etc.] by editing Step 1 in the `subroutine LASINPT`, file `"f10_taskinpt.f"`;

- set the normalized temporal power profile of every laser beam by editing the `function FLASTPRO` in file `"f10_taskinpt.f"`;
- select a model (or program a new one) for the normalized spatial power profile of every laser beam by setting a corresponding value of the `iwwprof` parameter in `function FLASSPRO` in file `"f10_taskinpt.f"`;

3. Boundary conditions and the initial state

Boundary conditions: `ibc` values, hydrodynamics, thermal conduction, radiation transport ...

Initial state: whether `pr0` or `tmp0`, when editing is needed ...

4. History plots

Data for history plots of various physical quantities are written into user specified files by calling the `subroutine TASKHIST` (file `"f12_taskout.f"`) from the output-master subroutine `HYDROOUT` (file `"f01_main.f"`); `TASKHIST` is called after every hydrocycle. All user-defined files for history plots are opened at the first call of `TASKHIST`. Thus, the entire `subroutine TASKHIST` is problem-specific, and usually must be programmed anew for each new problem.

5. Regular printout

Regular printout is written into the file `"pro.dat"` at user-defined time moments by calling the `subroutine TASKPRNT` (file `"f12_taskout.f"`) from the output-master subroutine `HYDROOUT` (file `"f01_main.f"`). The times for regular printout are governed by the user-defined array `twprnt(1:10)`. The values of `twprnt(1:10)` are assigned via the `namelist/input/`.

If `twprnt(3)` $\leq$ `twprnt(2)`, then printed are the regular-printout data from the hydrocycles ended at (or immediately after) the times

$$t = \texttt{twprnt(1)}, \texttt{twprnt(1)} + \texttt{twprnt(3)}, \texttt{twprnt(1)} + 2 * \texttt{twprnt(3)}, \ldots, \texttt{twprnt(2)};$$

regular printout is limited to times $t \leq$ `twprnt(2)`.

If `twprnt(3)` $>$ `twprnt(2)`, then printed are the regular-printout data from the hydrocycles ended at (or immediately after) the times

$$t = \texttt{twprnt(1)}, \texttt{twprnt(2)}, \texttt{twprnt(3)}, \ldots, \texttt{twprnt(10)}.$$

Example 1: `twprnt = 0.d0, 1.d0, 0.1d0` — the regular printout is performed with a time step $\Delta t = 0.1$ starting from time $t = 0$ until the time $t = 1$ has been reached.

Example 2: `twprnt = 0.1d0, 0.3d0, 0.35d0, 1.d100` — the regular printout is performed 3 times immediately after the time moments $t = 0.1$, 0.3, and 0.35.

To change the default set of the regular-printout data, one has to edit the `subroutine TASKPRNT`, and the two subroutines `PRNOUTC` (for printing out spatial profiles of cell-centered quantities) and `PRNOUTV` (for printing out spatial profiles of vertex-centered quantities) called from `TASKPRNT` — all in file `"f12_taskout.f"`.

7. Spectral image diagnostics

In the spectral diagnostics two different types of spectral images (namely, the *X-images* and the *R-graphs*) are written into corresponding output files at user-defined time moments by calling the spectral-output master routine `TASKXRSP` (file `"f12_taskout.f"`). The times for regular printout are governed by the user-defined array `twspctr(1:10)`, which obeys the same rules as the array `twprnt(1:10)` of times for regular printout. The values of `twspctr(1:10)` are assigned via the `namelist/input/`.

Example 1: `twspctr = 0.1d0, 0.3d0, 0.35d0, 1.d100` — the spectral output is performed 3 times immediately after the time moments $t = 0.1$, 0.3, and 0.35.

Example 2: `twspctr = 0.d0, 1.d0, 0.1d0` — the spectral output is performed with a time step $\Delta t = 0.1$ starting from time $t = 0$ until the time $t = 1$ has been reached.

Because the spectral radiation data are overwritten when passing to the next frequency, the spectral information is essentially lost by the end of each hydrocyle. Therefore, the spectral output occurs with a delay of one hydrocycle according to the following algorithm: once the criterion for this output is fulfilled (i.e. the time for the next spectral-image output is reached), the flag `ifwspctr` is set equal to 1 in the output-master subroutine `HYDROOUT` (file `"f01_main.f"`); then, in the next hydrocyle, the condition causes the master routine `TASKXRSP` for spectral images to be called from the radiation-transport master routine `RADDRV` (file `"f09_rad.f"`). When `ifwspctr=1`, the `subroutine TASKXRSP` is called for each frequency group from the frequency set `kfreqset` used for diagnostics (i.e. for the frequency set `kfreqset` = `nfreqsets`) but for only one primary photon propagation direction l = `iomega` = `iomeXimg`.

Depending on the value of the frequency-group counter `kfreq` at the moment when the master routine `TASKXRSP` is called, it performs different types of actions:

- if `TASKXRSP` is called with `kfreq` = 0, the initial operations are performed — like opening the necessary files, etc.;
- if `TASKXRSP` is called with $1 \leq$ `kfreq` ≤ 999999, the spectral output is prepared and written out for the frequency group `kfreq`;
- if `TASKXRSP` is called with `kfreq` ≥ 1000000, the final operations are performed — like closing all the spectral-output files, etc.

X-image spectral diagnostics. The *X-image* output is performed by the `subroutine WRXIMAGE` (file `"f12_taskout.f"`), called from the master routine `TASKXRSP`. The X-image data are written into two output files with names `"Ximg_nnn.dat"` and `"Ximg_totsp_nnn.dat"`, where `nnn` is the sequential number for the X-image output.

The X-image output uses exclusively the values of the group-integrated radiation intensity $I_{[k],i}$ at boundary nodes i in the frequency group k as calculated by the main RALEF algorithm used to solve the radiation transfer equation (i.e. obtained with the short-characteristic method) for one selected direction $\vec{\Omega}_{L_{Ximg}}$ from the total of $n(n/2+1)$ directions (over the 2π solid angle) treated in our version of the S_n method. The recording direction $\vec{\Omega}_{L_{Ximg}}$ is defined as the nearest to the user-specified X-image direction $\vec{\Omega}_{Ximg}$. It is identified by the values of variables `iocxXimg`, `iocyXimg`, `iomeXimg`, calculated in `subroutine RADSET`

(file "f09_rad.f") for the given $\vec{\Omega}_{Ximg}$. The latter is specified in Step 3 of the initialization subroutine TASKINPT (file "f10_taskinpt.f").

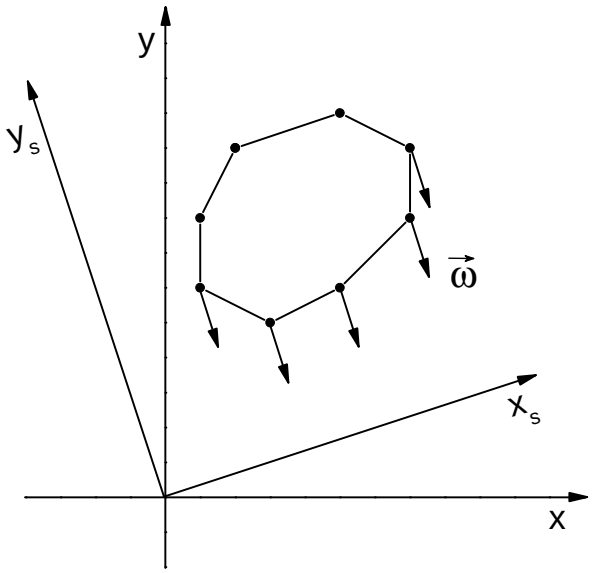


FIG. 16.8: Observation coordinates (x_s, y_s) for the X-image diagnostics; $\vec{\omega}$ is the projection of the photon propagation direction $\vec{\Omega}_{L_{Ximg}}$ onto the computational (x, y) plane.

The X-image is recorded in coordinates (x_s, y_s) obtained from the original (x, y) coordinates by a rotation transformation

$$\begin{cases} x_s = -\omega_y\, x + \omega_x\, y, \\ y_s = -\omega_x\, x - \omega_y\, y, \end{cases} \tag{16.77}$$

such that the x_s-axis is perpendicular to the propagation direction $\vec{\Omega}_{L_{Ximg}}$ of the registered X-rays, and the y_s-axis runs in the opposite to $\vec{\Omega}_{L_{Ximg}}$ direction (see Fig. 16.8); here $\vec{\omega} = (\omega_x, \omega_y)$ is the projection of the 3D vector $\vec{\Omega}_{L_{Ximg}}$ onto the computational (x, y) plane. The x_s-axis can be thought of as an imaginary observation slit. All the boundary vertices i visible through such a slit are ordered after the increasing values of their x_s coordinate, i.e. with $x_{s,1} \le x_{s,2} \le x_{s,3} \le \ldots$. Written into the file "Ximg_nnn.dat" are the values $x_{s,i}$, $y_{s,i}$ of the (x_s, y_s) coordinates of all the visible boundary vertices i, as well as the mean radiation intensities $I_{[k],i}/\Delta\nu_{[k]}$ for the photon propagation direction $\vec{\Omega}_{L_{Ximg}}$ at these vertices for all spectral groups $k = 1, 2, \ldots,$ nfreqs(nfreqsets).

File "Ximg_totsp_nnn.dat" provides data for a spatially integrated spectrum emitted in the direction $\vec{\Omega}_{L_{Ximg}}$. Namely, the integral spectral flux

$$F_{[k]} = \sum_{\text{visible } i} \frac{1}{2}(x_{s,i+1} - x_{s,i}) \frac{I_{[k],i} + I_{[k],i+1}}{\nu_{[k+1]} - \nu_{[k]}} \tag{16.78}$$

is calculated and written into the file "Ximg_totsp_nnn.dat" as a histogram function of $\nu_{[k]}$:

...	...	...
k	$\nu_{[k]}$	$F_{[k]}$
k	$\nu_{[k+1]}$	$F_{[k]}$
$k+1$	$\nu_{[k+1]}$	$F_{[k+1]}$
$k+1$	$\nu_{[k+2]}$	$F_{[k+1]}$
...	...	...

To set up the X-image diagnostics, the user only has to edit Step 3 in the `subroutine TASKINPT` (file "f10_taskinpt.f"), where the vector $\vec{\Omega}_{Ximg}$ is specified, and to verify that the `call WRXIMAGE` statements are not commented out in the `subroutine TASKXRSP`.

R-graph spectral diagnostics. The *R-graph* output is performed by the `subroutine WRRGRAPH` (file "f12_taskout.f"), called from the master routine `TASKXRSP`. The R-graph data are written into five output files with names "Rgr_img_nnn.dat", "Rgr_totsp_nnn.dat", "Rgr_od_nnn.dat", "Rgr_odc_nnn.dat" and "Rgr_tau_nnn.dat", where `nnn` is the sequential number for the R-graph output.

In contrast to the X-image diagnostics, the R-graph images are constructed independently of the main RALEF algorithm for radiation transport, by solving the radiation transfer equation in the post-processor mode along a user-defined set of rays (i.e. along long characteristics) traversing the computational domain. If two frequency sets have been specified from the start, then the spectral R-graph images are constructed for the second frequency set, which is independent (usually much more detailed) of the basic frequency set coupled to hydrodynamics.

The R-graph images are one-dimensional images along an imaginary observation slit for a prescribed photon propagation direction $\vec{\Omega}_{Rgr}$. The imaginary observation slit is specified by the 3D position of its origin point $\vec{x}_{Rgrsl}$ and the 3D vector $\vec{\Omega}_{Rgrsl}$ of its direction; see Fig. 16.9. Normally one chooses the vectors $\vec{\Omega}_{Rgrsl}$ and $\vec{\Omega}_{Rgr}$ to be perpendicular to one another.

An R-graph image for a given frequency group $[\nu_k, \nu_{k+1}]$ is constructed by integrating the transfer equation along a sequence of probe rays emanating from the ray-origin points along the imaginary observation slit. The number and the 1D distribution of the ray-origin points along the observation slit (i.e. over the distance s_{sl} along the slit measured from its origin point $\vec{x}_{Rgrsl}$) are defined by the user in the `subroutine WRRGRAPH` (file "f12_taskout.f"). To integrate the transfer equation along the probe rays, the ray tracing routine `RAYRUN` from the `module RAYTRACE` (file "f08_util.f") is used. The probe rays are traced either with a proper reflection from the radiation-reflecting boundaries (the flag `ifrayrfl=.true.`) or without such reflection (the flag `ifrayrfl=.false.`); the corresponding user-defined control flag `ifrayrfl` is set by editing Step 2.1 in the `subroutine WRRGRAPH` (file "f12_taskout.f"). The integration is performed from the starting point on the imaginary observation slit. Hence, if one wants to capture an R-graph image as seen by a distant observer, one has to ensure that the observation slit lies outside the physical computational region.

When integrating the transfer equation along a given probe ray for a given frequency group k, both the forward, $I^+_{[k]}$, (propagation direction $\vec{\Omega}_{Rgr}$ towards the observer), and the backward, $I^-_{[k]}$, (propagation direction $-\vec{\Omega}_{Rgr}$ away from the observer) group-integrated intensities are calculated as functions of the penetration depth s_{ray} along the ray; the distance s_{ray} is measured from the ray entrance point into the computational domain in the direction

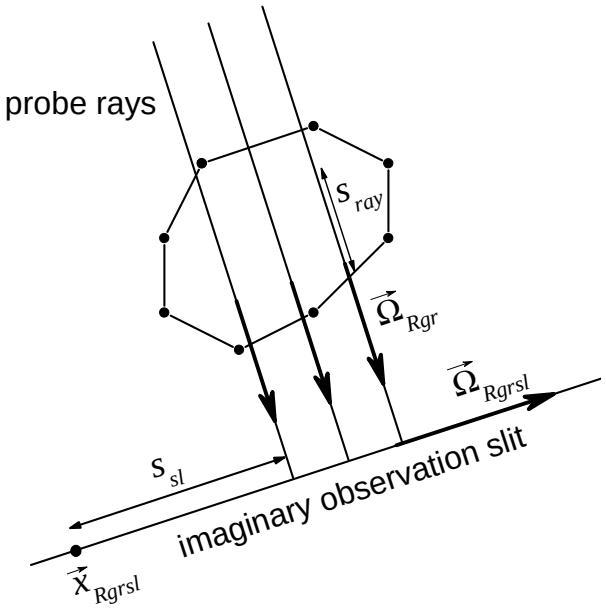


FIG. 16.9: Observation geometry for constructing the R-graph images.

$-\vec{\Omega}_{Rgr}$ away from the observation slit. The R-graph image is constructed by using the $I^+_{[k]}$ intensities on the observation slit line.

Written into the file `"Rgr_img_nnn.dat"` are the values of the coordinate s_{sl} along the slit for the probe ray origin points, and the observed mean forward intensities $I^+_{[k]}/\Delta_{[k]}$ at these points for each spectral group k. File `"Rgr_totsp_nnn.dat"` contains data for a spatially integrated spectrum (i.e. integrated along the slit coordinate s_{sl}) as observed through the slit in the form of a histogram plot $F_{[k]}$ versus $\nu_{[k]}$ — similar to the integrated spectrum in the X-image diagnostics. The integrated spectral flux $F_{[k]}$, calculated according to Eq. (16.78), is multiplied by an additional user-defined symmetry factor f_{sym}. Usually one has either $f_{sym} = 1$ or $f_{sym} = 2$; the latter value applies to situations where the probe rays cover only one half of a symmetric R-graph image, and one wants to have the total-spectrum data for the entire image. File `"Rgr_tau_nnn.dat"` lists the values of the optical depth $\tau_{[k]}$ in each frequency group k as a function of the penetration depth s_{ray} along the user-selected "central" ray with the number `krcww`.

Files `"Rgr_od_nnn.dat"` and `"Rgr_odc_nnn.dat"` provide information about the origin depth of radiation propagating along the probe rays towards the observation slit. The origin depth is defined as the penetration depth $s_{ray,od}$ where the ratio between the backward and the forward intensities is

$$\frac{I^-_{[k]}(s_{ray} = s_{ray,od})}{I^+_{[k]}(s_{ray} = s_{ray,od})} = \frac{1}{2}. \tag{16.79}$$

In optically thin plasmas this ratio corresponds to the mid-point along the ray trajectory across the plasma volume; in optically thick atmosphere-like configurations this ratio corresponds to the optical penetration depth of $\tau_{od} = \ln 2 = 0.69$. The file `"Rgr_od_nnn.dat"` contains the full set of the origin-depth data for all the probe rays and all the frequency groups, whereas the file `"Rgr_odc_nnn.dat"` contains the data for the "central" ray `krcww` only. Both files list the origin penetration depth $s_{ray,od}$, the x and y coordinates (x_{od}, y_{od}) of the origin point, the global CAVEAT cell number I_{od} of the origin point $[(x_{od}, y_{od}) \in$ cell $I_{od}]$, and the plasma density and temperature in cell I_{od}.

To set up the R-graph diagnostics, the user has to go through the following source-code editing steps

- edit Step 2 in the `subroutine TASKINPT`, file `"f10_taskinpt.f"`;
- edit Step 2.1 in the `subroutine WRRGRAPH`, file `"f12_taskout.f"`.

8. 2D visual graphics output (vtk-files)

Correspondence with the code variables:

$\Omega_{Ximg,x}$	= `omxXimg`	x-component of the user-defined vector $\vec{\Omega}_{Ximg}$ for the photon propagation direction in the X-image diagnostics;
$\Omega_{Ximg,y}$	= `omyXimg`	y-component of the user-defined vector $\vec{\Omega}_{Ximg}$;
$\Omega_{Ximg,z}$	= `omzXimg`	z-component of the user-defined vector $\vec{\Omega}_{Ximg}$;
$\Omega_{Rgr,x}$	= `OmxRgr`	x-component of the user-defined vector $\vec{\Omega}_{Rgr}$ for the photon propagation direction in the R-graph diagnostics;
$\Omega_{Rgr,y}$	= `OmyRgr`	y-component of the user-defined vector $\vec{\Omega}_{Rgr}$;
$\Omega_{Rgr,z}$	= `OmzRgr`	z-component of the user-defined vector $\vec{\Omega}_{Rgr}$;
x_{Rgrsl}	= `x0Rgrsl`	x-coordinate of the user-defined origin point $\vec{x}_{Rgrsl}$ on the imaginary observation slit in the R-graph diagnostics;
y_{Rgrsl}	= `y0Rgrsl`	y-coordinate of the user-defined origin point $\vec{x}_{Rgrsl}$ on the observation slit in the R-graph diagnostics;
z_{Rgrsl}	= `z0Rgrsl`	z-coordinate of the user-defined origin point $\vec{x}_{Rgrsl}$ on the observation slit in the R-graph diagnostics;
$\Omega_{Rgrsl,x}$	= `OmRgrslx`	x-component of the user-defined vector $\vec{\Omega}_{Rgrsl}$ along the imaginary observation slit in the R-graph diagnostics;
$\Omega_{Rgrsl,y}$	= `OmRgrsly`	y-component of the user-defined vector $\vec{\Omega}_{Rgrsl}$ along the observation slit in the R-graph diagnostics;
$\Omega_{Rgr,z}$	= `OmRgrslz`	z-component of the user-defined vector $\vec{\Omega}_{Rgrsl}$ along the observation slit in the R-graph diagnostics;
f_{sym}	= `symfactor`	symmetry factor for writing out the spatially integrated spectrum; allowed values are $f_{sym} = 1$ or $f_{sym} = 2$;

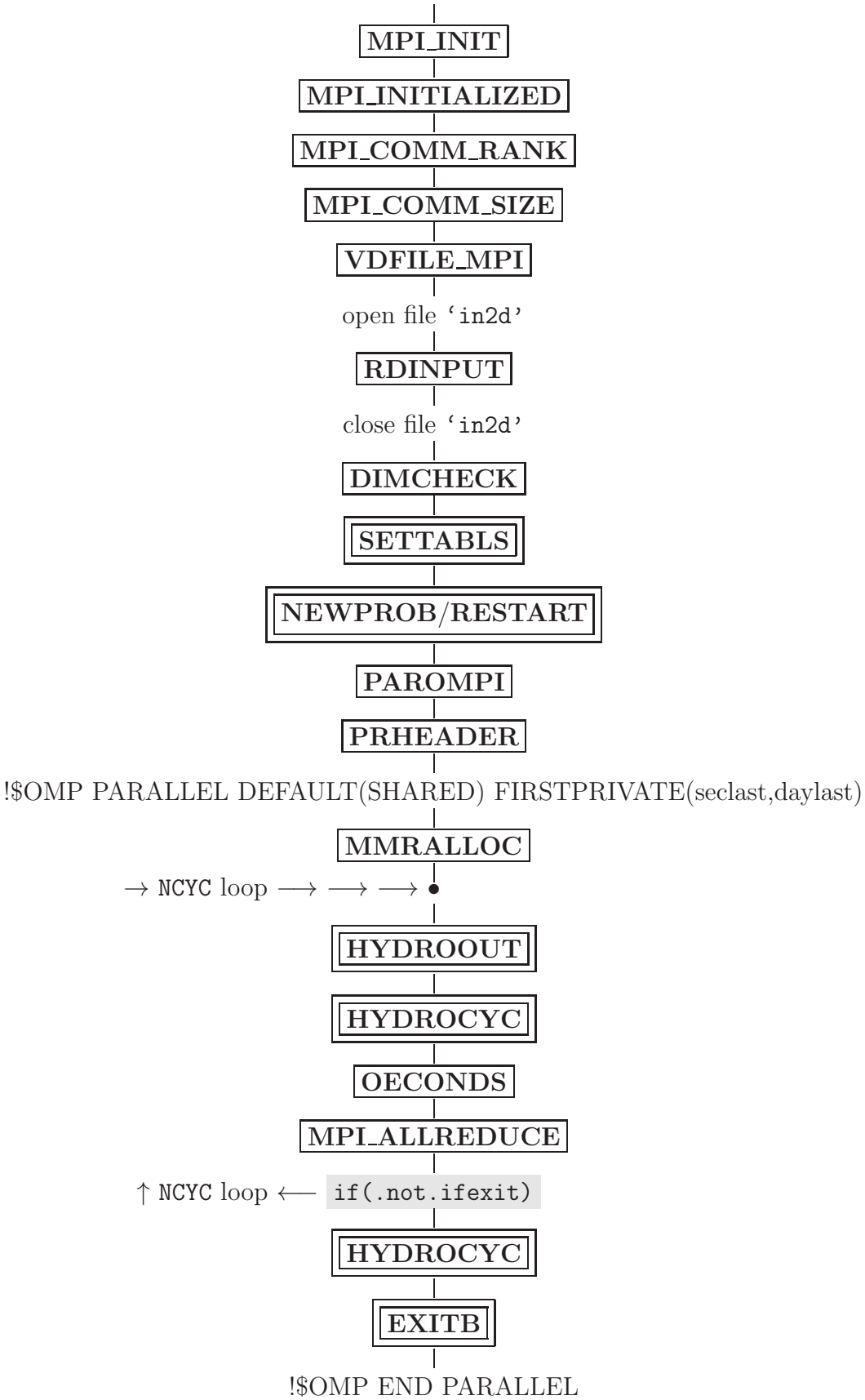
[1] F. L. Addessio, J. R. Baumgardner, J. K. Dukowicz, N. L. Johnson, B. A. Kashiwa, R. M. Rauenzahn, and C. Zemach, Report LA-10613-MS-Rev. 1, UC-32, Los Alamos National Laboratory (1992).

[2] M. M. Basko, J. A. Maruhn, and A. Tauschwitz, Journal of Computational Physics **228**, 2175 (2009), ISSN 0021-9991, URL `http://www.sciencedirect.com/science/article/pii/S0021999108006232`.

APPENDIX A: FLOWCHARTS OF SUBROUTINE CALLS AND MEMORY ALLOCATION

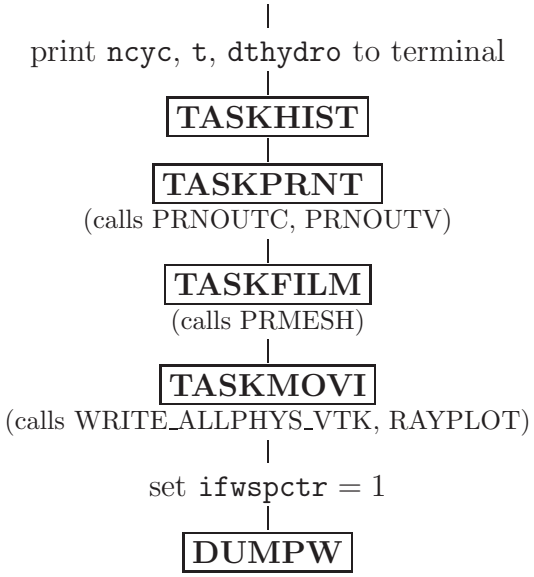
For complex subroutines put in a double frame, like **ABC**, separate flowcharts of their own are presented. When several subroutine names are combined in a single frame, it means that they are called in a loop over mesh blocks (in an `iblk` loop).

Program RALEF:

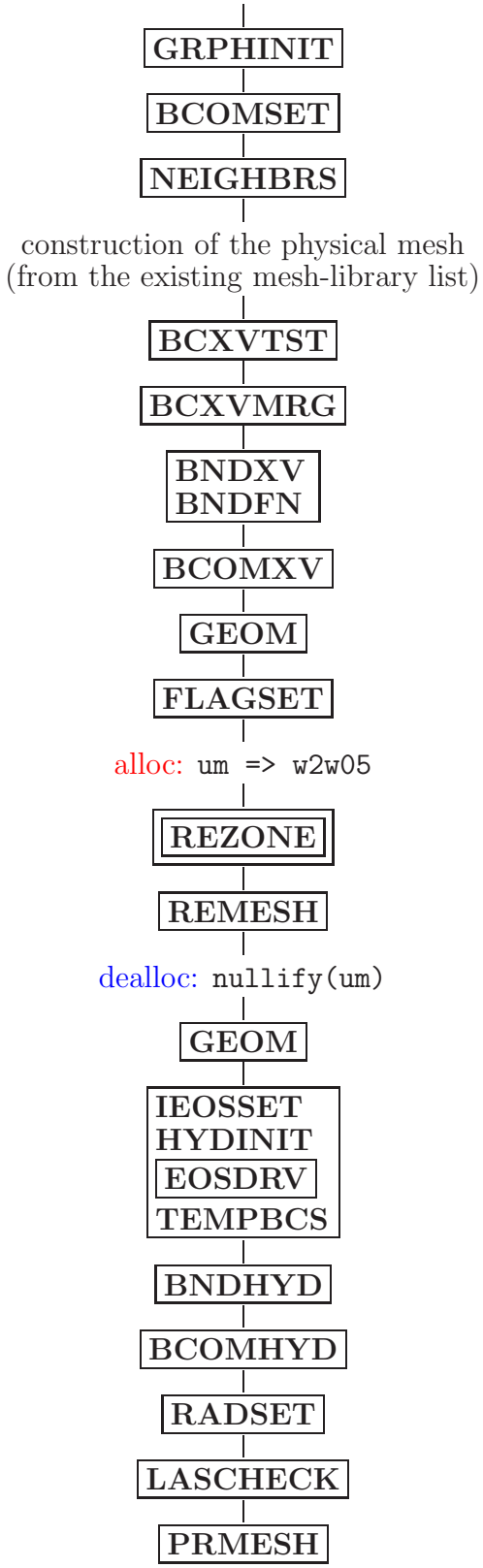


Subroutine SETTABLS:

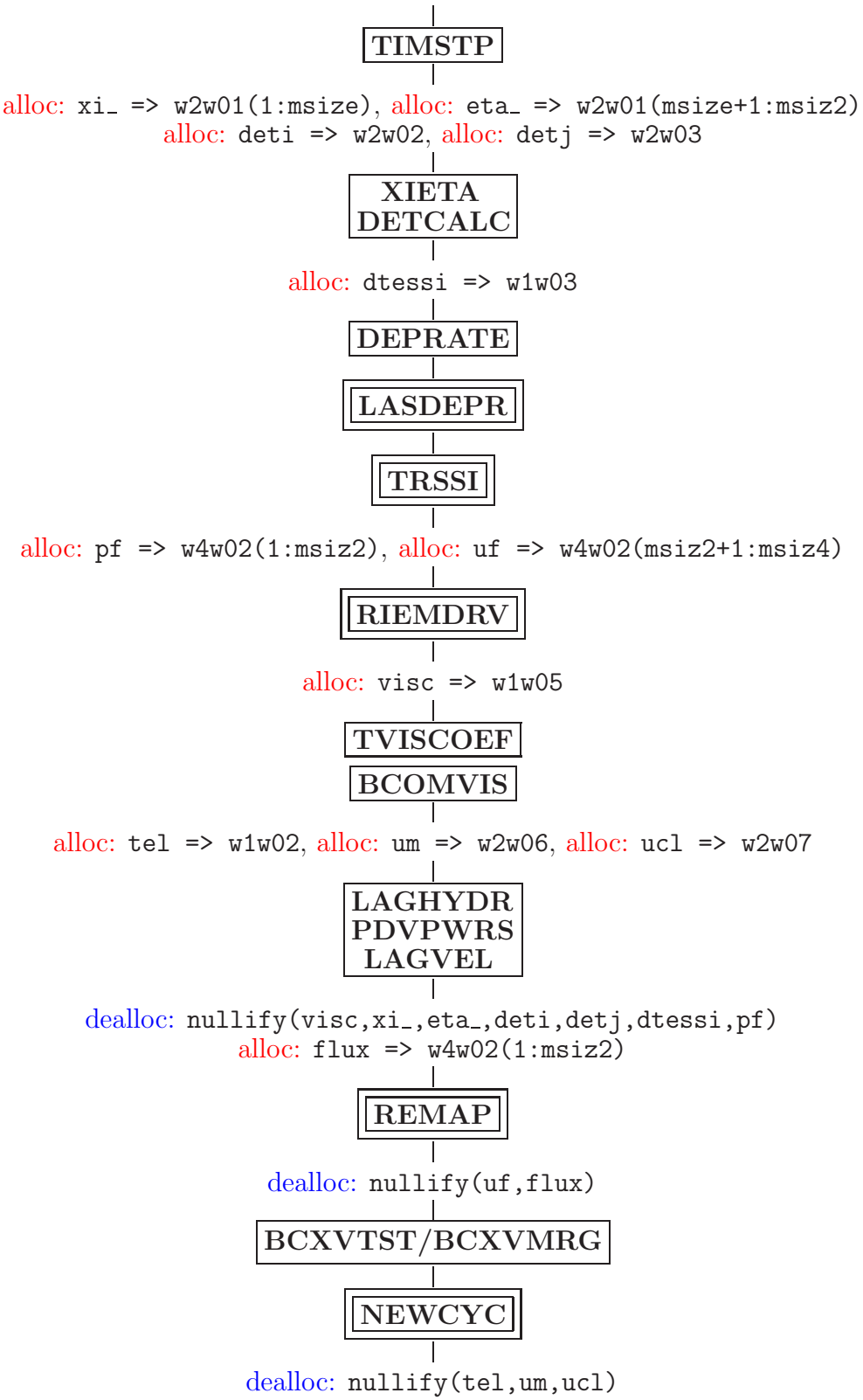
Subroutine HYDROOUT:



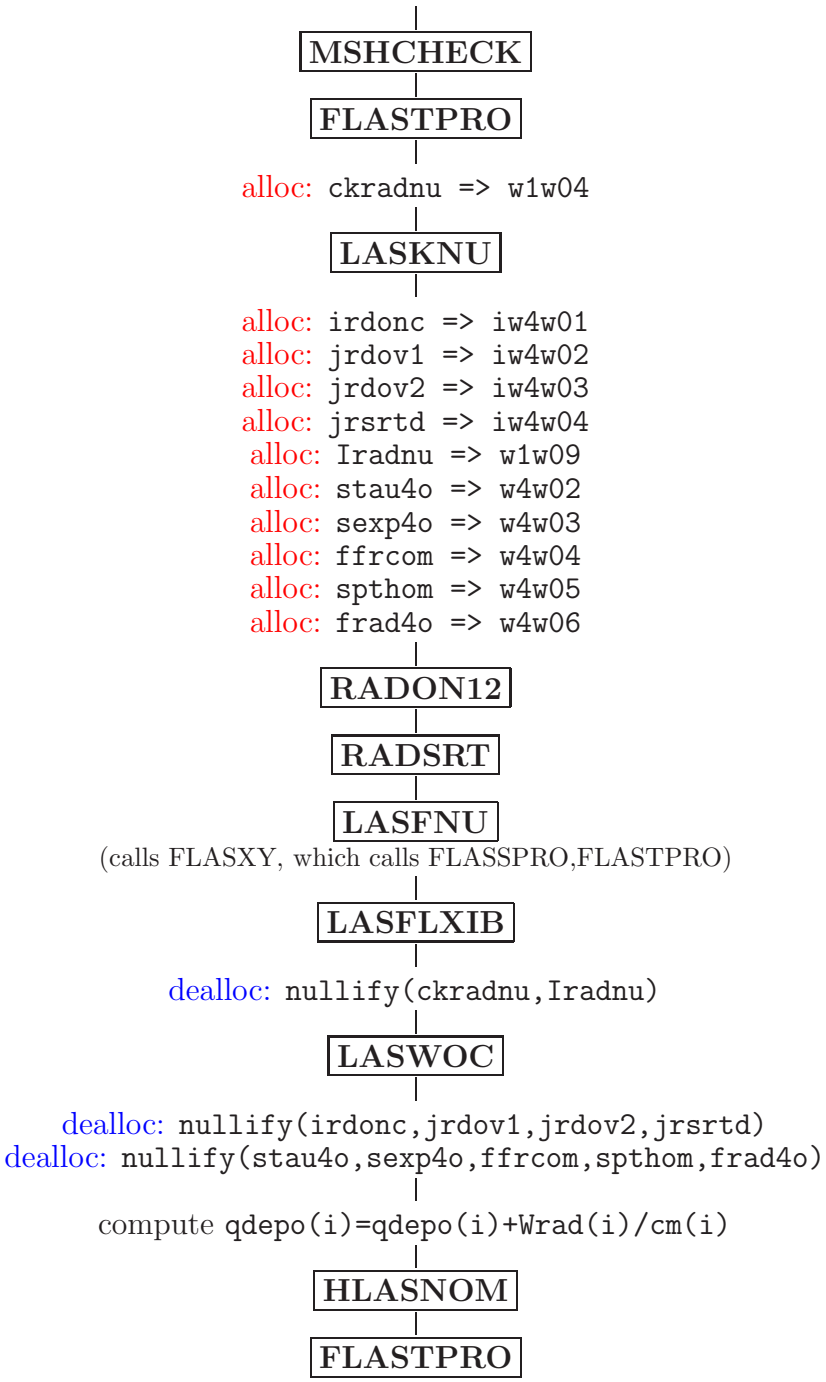
Subroutine NEWPROB:



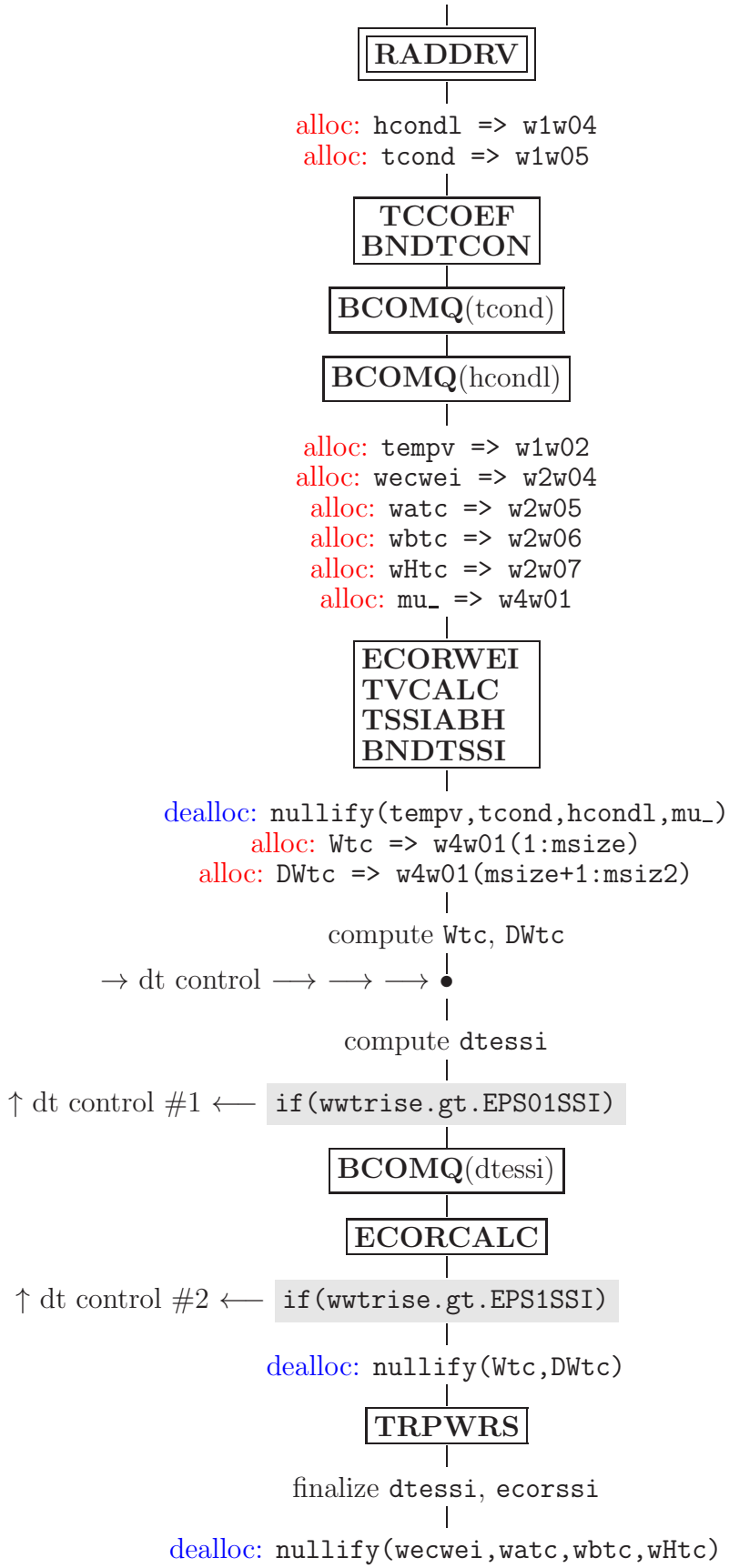
Subroutine HYDROCYC:



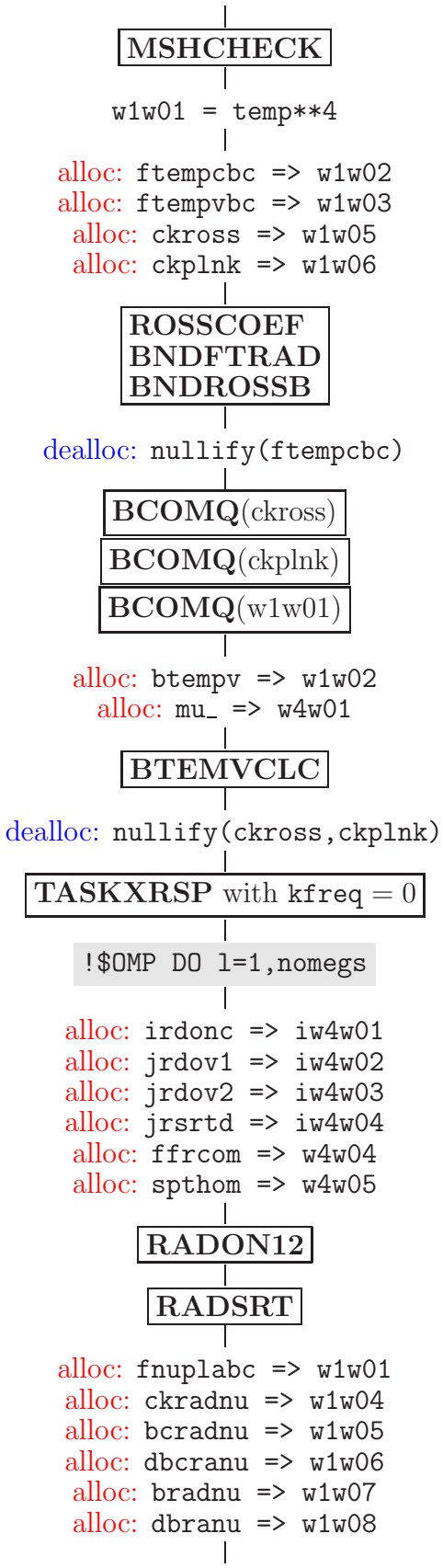
Subroutine LASDEPR:

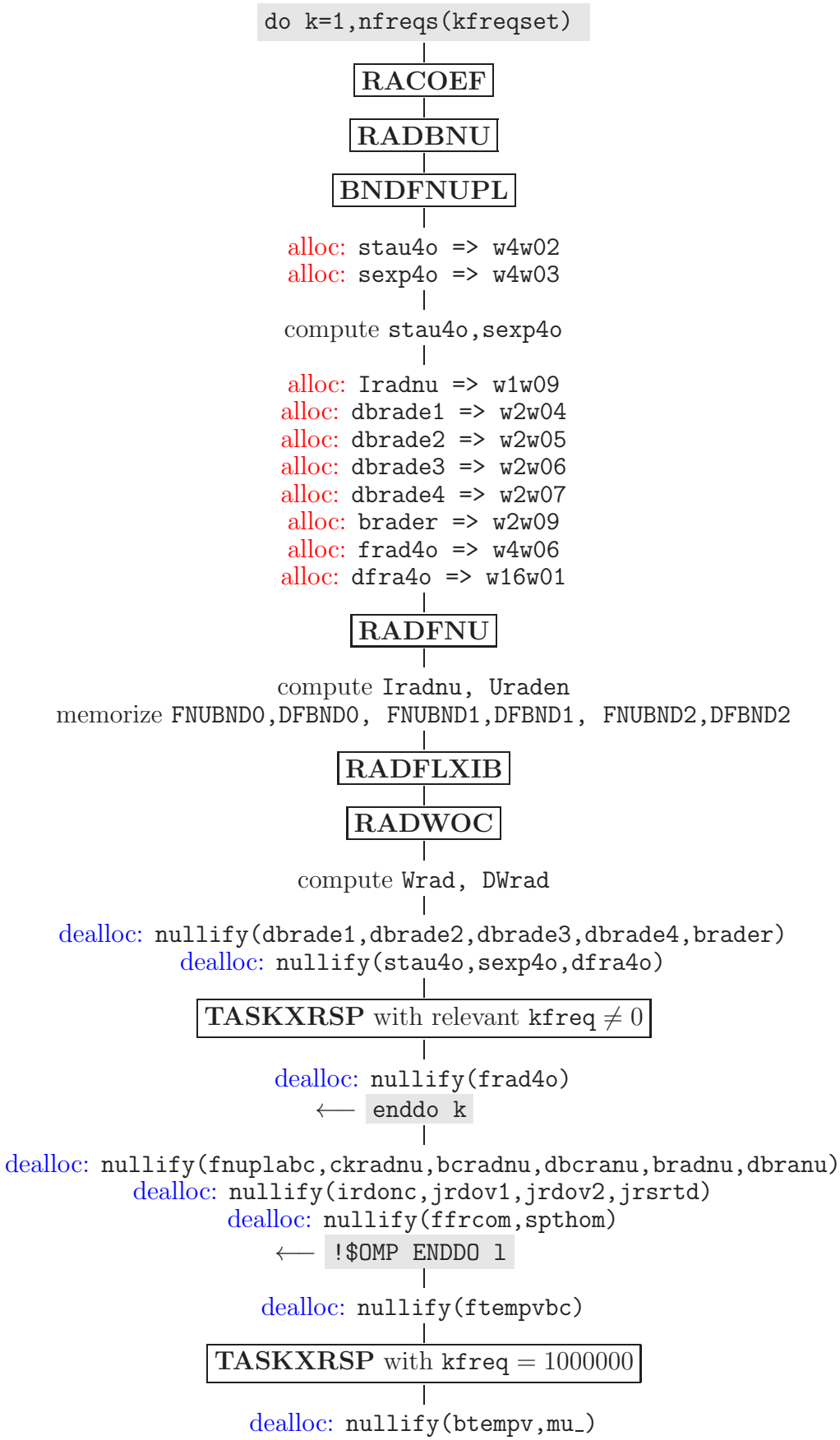


Subroutine TRSSI:

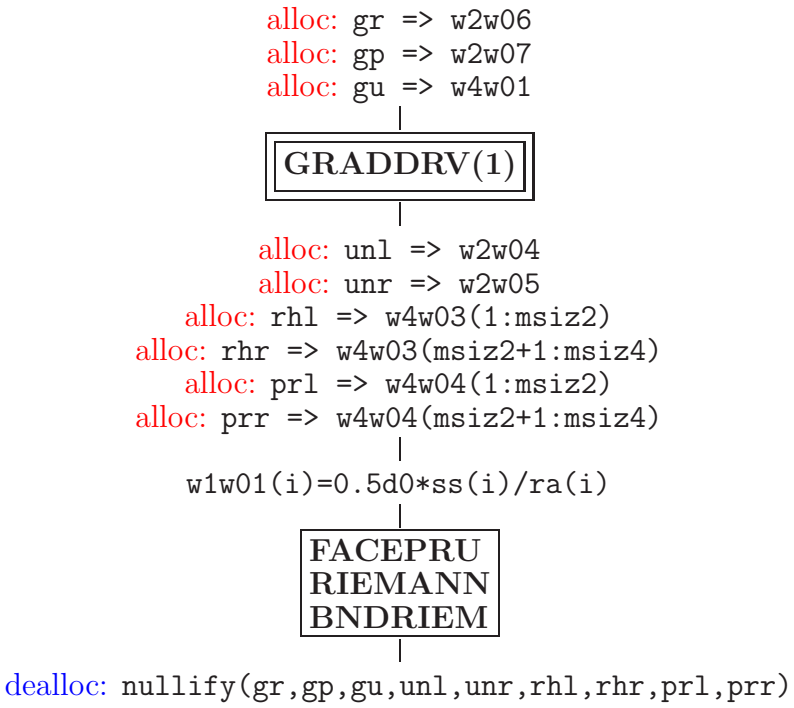


Subroutine RADDRV:

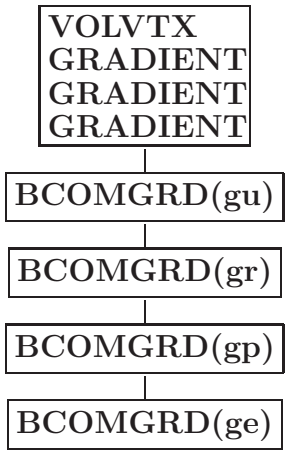



do k=1,nfreqs(kfreqset)
RACOEF
RADBNU
BNDFNUPL
alloc: stau4o => w4w02
alloc: sexp4o => w4w03
compute stau4o,sexp4o
alloc: Iradnu => w1w09
alloc: dbrade1 => w2w04
alloc: dbrade2 => w2w05
alloc: dbrade3 => w2w06
alloc: dbrade4 => w2w07
alloc: brader => w2w09
alloc: frad4o => w4w06
alloc: dfra4o => w16w01
RADFNU
compute Iradnu, Uraden
memorize FNUBND0,DFBND0, FNUBND1,DFBND1, FNUBND2,DFBND2
RADFLXIB
RADWOC
compute Wrad, DWrad
dealloc: nullify(dbrade1,dbrade2,dbrade3,dbrade4,brader)
dealloc: nullify(stau4o,sexp4o,dfra4o)
TASKXRSP with relevant kfreq ≠ 0
dealloc: nullify(frad4o)
⟵ enddo k
dealloc: nullify(fnuplabc,ckradnu,bcradnu,dbcranu,bradnu,dbranu)
dealloc: nullify(irdonc,jrdov1,jrdov2,jrsrtd)
dealloc: nullify(ffrcom,spthom)
⟵ !$OMP ENDDO 1
dealloc: nullify(ftempvbc)
TASKXRSP with kfreq = 1000000
dealloc: nullify(btempv,mu_)

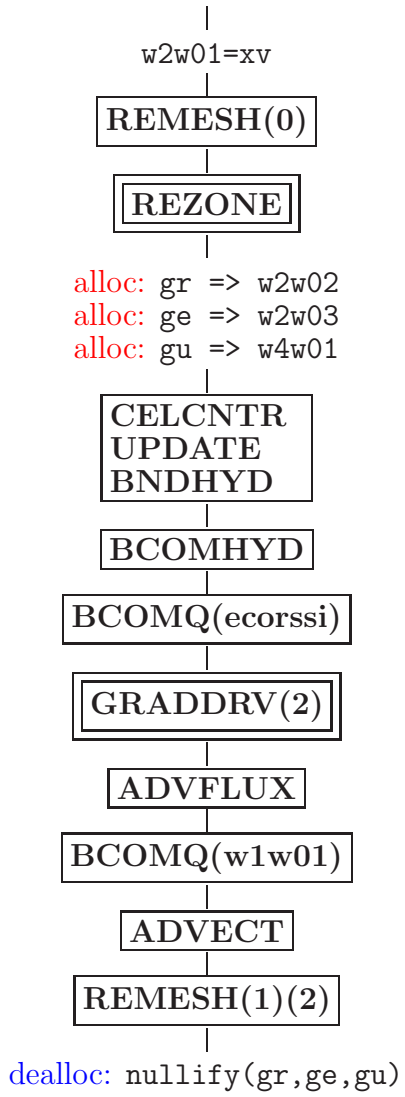
Subroutine RIEMDRV:



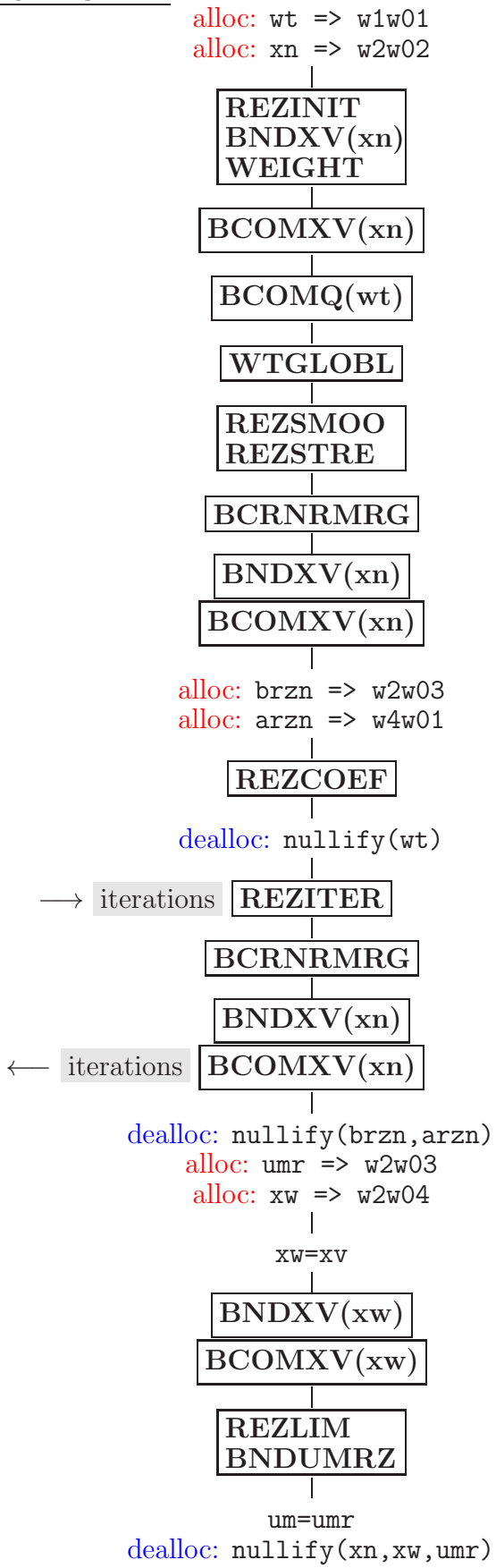
Subroutine GRADDRV:



Subroutine REMAP:



Subroutine REZONE:



Subroutine NEWCYC: